

Parallel Algorithms for Big Data Optimization

Francisco Facchinei, Simone Sagratella, and Gesualdo Scutari *Senior Member, IEEE*

Abstract—We propose a decomposition framework for the parallel optimization of the sum of a differentiable function and a (block) separable nonsmooth, convex one. The latter term is usually employed to enforce structure in the solution, typically sparsity. Our framework is very flexible and includes both fully parallel Jacobi schemes and Gauss-Seidel (i.e., sequential) ones, as well as virtually all possibilities “in between” with only a subset of variables updated at each iteration. Our theoretical convergence results improve on existing ones, and numerical results on LASSO and logistic regression problems show that the new method consistently outperforms existing algorithms.

Index Terms—Parallel optimization, Distributed methods, Jacobi method, LASSO, Sparse solution.

I. INTRODUCTION

The minimization of the sum of a smooth function, F , and of a nonsmooth (block separable) convex one, G ,

$$\min_{\mathbf{x} \in X} V(\mathbf{x}) \triangleq F(\mathbf{x}) + G(\mathbf{x}), \quad (1)$$

is an ubiquitous problem that arises in many fields of engineering, so diverse as compressed sensing, basis pursuit denoising, sensor networks, neuroelectromagnetic imaging, machine learning, data mining, sparse logistic regression, genomics, meteorology, tensor factorization and completion, geophysics, and radio astronomy. Usually the nonsmooth term is used to promote sparsity of the optimal solution, which often corresponds to a parsimonious representation of some phenomenon at hand. Many of the aforementioned applications can give rise to extremely large problems so that standard optimization techniques are hardly applicable. And indeed, recent years have witnessed a flurry of research activity aimed at developing solution methods that are simple (for example based solely on matrix/vector multiplications) but yet capable to converge to a good approximate solution in reasonable time. It is hardly possible here to even summarize the huge amount of work done in this field; we refer the reader to the recent works [2]–[17] and books [18]–[20] as entry points to the literature.

However, with big data problems it is clearly necessary to design *parallel* methods able to exploit the computational power of multi-core processors in order to solve many interesting problems. It is then surprising that while sequential solutions methods for Problem (1) have been widely investigated, the analysis of parallel algorithms suitable to large-scale

implementations lags behind. Gradient-type methods can of course be easily parallelized, but they are known to generally suffer from slow convergence; furthermore, by linearizing F they do not exploit any structure of F , a fact that instead has been shown to enhance convergence speed [21]. However, beyond that, and looking at recent approaches, we are only aware of very few papers that deal with parallel solution methods [9]–[16]. These papers analyze both randomized and deterministic block Coordinate Descent Methods (CDMs); one advantage of the analyses therein is that they provide an interesting (global) rate of convergence. However, i) they are essentially still (regularized) gradient-based methods; ii) they are not flexible enough to include, among other things, very natural Jacobi-type methods (where at each iteration a minimization of the original function is performed *in parallel* with respect to *all* blocks of variables); and iii) except for [10], [11], [13], they cannot deal with a nonconvex F . We refer to Section V for a detailed discussion on current parallel and sequential solution methods for (1).

In this paper, we propose a new, broad, deterministic algorithmic framework for the solution of Problem (1). The essential, rather natural idea underlying our approach is to decompose (1) into a sequence of (simpler) subproblems whereby the function F is replaced by suitable convex approximations; the subproblems can be solved in a *parallel* and *distributed* fashion. Key (new) features of the proposed algorithmic framework are: i) it is parallel, with a degree of parallelism that can be chosen by the user and that can go from a complete parallelism (every variable is updated in parallel to all the others) to the sequential (only one variable is updated at each iteration), covering virtually all the possibilities in “between”; ii) it easily leads to distributed implementations; iii) it can tackle a nonconvex F ; iv) it is very flexible and includes, among others, updates based on gradient- or Newton-type approximations; v) it easily allows for inexact solution of the subproblems; vi) it permits the update of only some (blocks of) variables at each iteration (a feature that turns out to be very important numerically); vii) even in the case of the minimization of a smooth, convex function (i.e., $F \in C^1$ is convex and $G \equiv 0$) our theoretical results compare favorably to state-of-the-art methods.

The proposed framework encompasses a gamut of novel algorithms, offering a lot of flexibility to control iteration complexity, communication overhead, and convergence speed, while *converging under the same conditions*; these desirable features make our schemes applicable to several different problems and scenarios. Among the variety of new updating rules for the (block) variables we propose, it is worth mentioning a combination of Jacobi and Gauss-Seidel updates, which seems particularly valuable in parallel optimization on multi core/processor architectures; to the best of our knowledge this is the first time that such a scheme is proposed and analyzed.

A further contribution of the paper is to implement our

The order of the authors is alphabetic; all the authors contributed equally to the paper.

F. Facchinei and S. Sagratella are with the Dept. of Computer, Control, and Management Engineering, at Univ. of Rome La Sapienza, Rome, Italy. Emails: <facchinei,sagratella>@dis.uniroma1.it

G. Scutari is with the Dept. of Electrical Engineering, at the State Univ. of New York at Buffalo, Buffalo, USA. Email: gesualdo@buffalo.edu. His work was supported by the USA National Science Foundation under Grants CMS 1218717 and CAREER Award No. 1254739.

Part of this work has been presented at the 2014 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2014), Florence, Italy, May 4-9 2014, [1].

schemes and the most representative ones in the literature over a parallel architecture, the General Compute Cluster of the Center for Computational Research at the State University of New York at Buffalo. Numerical results on LASSO and Logistic Regression problems show that our algorithms consistently outperform state-of-the-art schemes.

The paper is organized as follows. Section II formally introduces the optimization problem along with the main assumptions under which it is studied. Section III describes our novel general algorithmic framework along with its convergence properties. In Section IV we discuss several instances of the general scheme introduced in Section III. Section V contains a detailed comparison of our schemes with state-of-the-art algorithms on similar problems. Numerical results are presented in Section VI, where we focus on LASSO and Logistic Regression problems and compare our schemes with state-of-the-art alternative solution methods. Finally, Section VII draws some conclusions. All proofs of our results are given in the Appendix.

II. PROBLEM DEFINITION

We consider Problem (1), where the feasible set $X = X_1 \times \dots \times X_N$ is a Cartesian product of lower dimensional convex sets $X_i \subseteq \mathbb{R}^{n_i}$, and $\mathbf{x} \in \mathbb{R}^n$ is partitioned accordingly: $\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$, with each $\mathbf{x}_i \in \mathbb{R}^{n_i}$; F is smooth (and not necessarily convex) and G is convex and possibly nondifferentiable, with $G(\mathbf{x}) = \sum_{i=1}^N g_i(\mathbf{x}_i)$. This formulation is very general and includes problems of great interest. Below we list some instances of Problem (1).

- $G(\mathbf{x}) = 0$; in this case the problem reduces to the minimization of a smooth, possibly nonconvex problem with convex constraints.
- $F(\mathbf{x}) = \|\mathbf{Ax} - \mathbf{b}\|^2$ and $G(\mathbf{x}) = c\|\mathbf{x}\|_1$, $X = \mathbb{R}^n$, with $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$, and $c \in \mathbb{R}_{++}$ given constants; this is the renowned and much studied LASSO problem [2].
- $F(\mathbf{x}) = \|\mathbf{Ax} - \mathbf{b}\|^2$ and $G(\mathbf{x}) = c \sum_{i=1}^N \|\mathbf{x}_i\|_2$, $X = \mathbb{R}^n$, with $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$, and $c \in \mathbb{R}_{++}$ given constants; this is the group LASSO problem [22].
- $F(\mathbf{x}) = \sum_{j=1}^m \log(1 + e^{-a_j \mathbf{y}_j^T \mathbf{x}})$ and $G(\mathbf{x}) = c\|\mathbf{x}\|_1$ (or $G(\mathbf{x}) = c \sum_{i=1}^N \|\mathbf{x}_i\|_2$), with $\mathbf{y}_j \in \mathbb{R}^n$, $a_j \in \mathbb{R}$, and $c \in \mathbb{R}_{++}$ given constants; this is the sparse logistic regression problem [23], [24].
- $F(\mathbf{x}) = \sum_{j=1}^m \max\{0, 1 - a_j \mathbf{y}_j^T \mathbf{x}\}^2$ and $G(\mathbf{x}) = c\|\mathbf{x}\|_1$, with $a_j \in \{-1, 1\}$, $\mathbf{y}_j \in \mathbb{R}^n$, and $c \in \mathbb{R}_{++}$ given; this is the ℓ_1 -regularized ℓ_2 -loss Support Vector Machine problem [5].
- Other problems that can be cast in the form (1) include the Nuclear Norm Minimization problem, the Robust Principal Component Analysis problem, the Sparse Inverse Covariance Selection problem, the Nonnegative Matrix (or Tensor) Factorization problem, see e.g., [25] and references therein.

Assumptions. Given (1), we make the following blanket assumptions:

- (A1) Each X_i is nonempty, closed, and convex;
- (A2) F is C^1 on an open set containing X ;
- (A3) ∇F is Lipschitz continuous on X with constant L_F ;
- (A4) $G(\mathbf{x}) = \sum_{i=1}^N g_i(\mathbf{x}_i)$, with all g_i continuous and convex on X_i ;

(A5) V is coercive.

Note that the above assumptions are standard and are satisfied by most of the problems of practical interest. For instance, A3 holds automatically if X is bounded; the block-separability condition A4 is a common assumption in the literature of parallel methods for the class of problems (1) (it is in fact instrumental to deal with the nonsmoothness of G in a parallel environment). Interestingly A4 is satisfied by all standard G usually encountered in applications, including $G(\mathbf{x}) = \|\mathbf{x}\|_1$ and $G(\mathbf{x}) = \sum_{i=1}^N \|\mathbf{x}_i\|_2$, which are among the most commonly used functions. Assumption A5 is needed to guarantee that the sequence generated by our method is bounded; we could dispense with it at the price of a more complex analysis and cumbersome statement of convergence results.

III. MAIN RESULTS

We begin introducing an informal description of our algorithmic framework along with a list of key features that we would like our schemes enjoy; this will shed light on the core idea of the proposed decomposition technique.

We want to develop *parallel* solution methods for Problem (1) whereby operations can be carried out on some or (possibly) all (block) variables \mathbf{x}_i at the *same* time. The most natural parallel (Jacobi-type) method one can think of is updating *all* blocks simultaneously: given \mathbf{x}^k , each (block) variable \mathbf{x}_i is updated by solving the following subproblem

$$\mathbf{x}_i^{k+1} \in \operatorname{argmin}_{\mathbf{x}_i \in X_i} \{F(\mathbf{x}_i, \mathbf{x}_{-i}^k) + g_i(\mathbf{x}_i)\}, \quad (2)$$

where \mathbf{x}_{-i} denotes the vector obtained from \mathbf{x} by deleting the block \mathbf{x}_i . Unfortunately this method converges only under very restrictive conditions [26] that are seldom verified in practice. To cope with this issue the proposed approach introduces some “memory” in the iterate: the new point is a convex combination of \mathbf{x}^k and the solutions of (2). Building on this iterate, we would like our framework to enjoy many additional features, as described next.

Approximating F : Solving each subproblem as in (2) may be too costly or difficult in some situations. One may then prefer to approximate this problem, in some suitable sense, in order to facilitate the task of computing the new iteration. To this end, we assume that for all $i \in \mathcal{N} \triangleq \{1, \dots, N\}$ we can define a function $P_i(\mathbf{z}; \mathbf{w}) : X_i \times X \rightarrow \mathbb{R}$, the candidate approximant of F , having the following properties (we denote by ∇P_i the partial gradient of P_i with respect to the first argument \mathbf{z}):

- (P1) $P_i(\bullet; \mathbf{w})$ is convex and continuously differentiable on X_i for all $\mathbf{w} \in X$;
- (P2) $\nabla P_i(\mathbf{x}_i; \mathbf{x}) = \nabla_{\mathbf{x}_i} F(\mathbf{x})$ for all $\mathbf{x} \in X$;
- (P3) $\nabla P_i(\mathbf{z}; \bullet)$ is Lipschitz continuous on X for all $\mathbf{z} \in X_i$.

Such a function P_i should be regarded as a (simple) convex approximation of F at the point \mathbf{x} with respect to the block of variables \mathbf{x}_i that preserves the first order properties of F with respect to \mathbf{x}_i .

Based on this approximation we can define at any point $\mathbf{x}^k \in X$ a *regularized* approximation $\tilde{h}_i(\mathbf{x}_i; \mathbf{x}^k)$ of V with respect to \mathbf{x}_i wherein F is replaced by P_i while the nondifferentiable term is preserved, and a quadratic proximal term

is added to make the overall approximation strongly convex. More formally, we have

$$\begin{aligned} \tilde{h}_i(\mathbf{x}_i; \mathbf{x}^k) &\triangleq \underbrace{P_i(\mathbf{x}_i; \mathbf{x}^k) + \frac{\tau_i}{2} (\mathbf{x}_i - \mathbf{x}_i^k)^T \mathbf{Q}_i(\mathbf{x}^k) (\mathbf{x}_i - \mathbf{x}_i^k)}_{\triangleq h_i(\mathbf{x}_i; \mathbf{x}^k)} \\ &\quad + g_i(\mathbf{x}_i), \end{aligned}$$

where $\mathbf{Q}_i(\mathbf{x}^k)$ is an $n_i \times n_i$ positive definite matrix (possibly dependent on \mathbf{x}^k). We always assume that the functions $h_i(\bullet; \mathbf{x}_i^k)$ are uniformly strongly convex.

(A6) All $h_i(\bullet; \mathbf{x}^k)$ are uniformly strongly convex on X_i with a common positive definiteness constant $q > 0$; furthermore, $\mathbf{Q}_i(\bullet)$ is Lipschitz continuous on X .

Note that an easy and standard way to satisfy A6 is to take, for any i and for any k , $\tau_i = q > 0$ and $\mathbf{Q}_i(\mathbf{x}^k) = \mathbf{I}$. However, if $P_i(\bullet; \mathbf{x}^k)$ is already uniformly strongly convex, one can avoid the proximal term and set $\tau_i = 0$ while satisfying A6.

Associated with each i and point $\mathbf{x}^k \in X$ we can define the following optimal block solution map:

$$\hat{\mathbf{x}}_i(\mathbf{x}^k, \tau_i) \triangleq \operatorname{argmin}_{\mathbf{x}_i \in X_i} \tilde{h}_i(\mathbf{x}_i; \mathbf{x}^k). \quad (3)$$

Note that $\hat{\mathbf{x}}_i(\mathbf{x}^k, \tau_i)$ is always well-defined, since the optimization problem in (3) is strongly convex. Given (3), we can then introduce the solution map

$$X \ni \mathbf{y} \mapsto \hat{\mathbf{x}}(\mathbf{y}, \boldsymbol{\tau}) \triangleq (\hat{\mathbf{x}}_i(\mathbf{y}, \tau_i))_{i=1}^N.$$

The proposed algorithm (that we formally describe later on) is based on the computation of (an approximation of) $\hat{\mathbf{x}}(\mathbf{x}^k, \boldsymbol{\tau})$. Therefore the functions P_i should lead to as easily computable functions $\hat{\mathbf{x}}$ as possible. An appropriate choice depends on the problem at hand and on computational requirements. We discuss alternative possible choices for the approximations P_i in Section IV.

Inexact solutions: In many situations (especially in the case of large-scale problems), it can be useful to further reduce the computational effort needed to solve the subproblems in (3) by allowing *inexact* computations \mathbf{z}^k of $\hat{\mathbf{x}}_i(\mathbf{x}^k, \tau_i)$, i.e., $\|\mathbf{z}_i^k - \hat{\mathbf{x}}_i(\mathbf{x}^k, \tau_i)\| \leq \varepsilon_i^k$, where ε_i^k measures the accuracy in computing the solution.

Updating only some blocks: Another important feature we want for our algorithmic framework is the capability of updating at each iteration only *some* of the (block) variables, a feature that has been observed to be very effective numerically. In fact, our schemes are guaranteed to converge under the update of only a *subset* of the variables at each iteration; the only condition is that such a subset contains at least one (block) component which is within a factor $\rho \in (0, 1]$ “far away” from the optimality, in the sense explained next. Since \mathbf{x}_i^k is an optimal solution of (3) if and only if $\hat{\mathbf{x}}_i(\mathbf{x}^k, \tau_i) = \mathbf{x}_i^k$, a natural distance of \mathbf{x}_i^k from the optimality is $d_i^k \triangleq \|\hat{\mathbf{x}}_i(\mathbf{x}^k, \tau_i) - \mathbf{x}_i^k\|$; one could then select the blocks \mathbf{x}_i ’s to update based on such an optimality measure (e.g., opting for blocks exhibiting larger d_i^k ’s). However, this choice requires the computation of all the solutions $\hat{\mathbf{x}}_i(\mathbf{x}^k, \tau_i)$, for $i = 1, \dots, n$, which in some applications (e.g., huge-scale problems) might be computationally too expensive. Building

on the same idea, we can introduce alternative less expensive metrics by replacing the distance $\|\hat{\mathbf{x}}_i(\mathbf{x}^k, \tau_i) - \mathbf{x}_i^k\|$ with a computationally cheaper *error bound*, i.e., a function $E_i(\mathbf{x})$ such that

$$\underline{s}_i \|\hat{\mathbf{x}}_i(\mathbf{x}^k, \tau_i) - \mathbf{x}_i^k\| \leq E_i(\mathbf{x}^k) \leq \bar{s}_i \|\hat{\mathbf{x}}_i(\mathbf{x}^k, \tau_i) - \mathbf{x}_i^k\|, \quad (4)$$

for some $0 < \underline{s}_i \leq \bar{s}_i$. Of course one can always set $E_i(\mathbf{x}^k) = \|\hat{\mathbf{x}}_i(\mathbf{x}^k, \tau_i) - \mathbf{x}_i^k\|$, but other choices are also possible; we discuss this point further in Section IV.

Algorithmic framework: We are now ready to formally introduce our algorithm, Algorithm 1, that includes all the features discussed above; convergence to stationary solutions¹ of (1) is stated in Theorem 1.

Algorithm 1: Inexact Flexible Parallel Algorithm (FLEXA)

Data : $\{\varepsilon_i^k\}$ for $i \in \mathcal{N}$, $\boldsymbol{\tau} \geq \mathbf{0}$, $\{\gamma^k\} > 0$, $\mathbf{x}^0 \in X$, $\rho \in (0, 1]$.
Set $k = 0$.

(S.1) : If \mathbf{x}^k satisfies a termination criterion: STOP;

(S.2) : For all $i \in \mathcal{N}$, solve (3) with accuracy ε_i^k :
Find $\mathbf{z}_i^k \in X_i$ s.t. $\|\mathbf{z}_i^k - \hat{\mathbf{x}}_i(\mathbf{x}^k, \tau_i)\| \leq \varepsilon_i^k$;

(S.3) : Set $M^k \triangleq \max_i \{E_i(\mathbf{x}^k)\}$.

Choose a set S^k that contains at least one index i for which $E_i(\mathbf{x}^k) \geq \rho M^k$.

Set $\hat{\mathbf{z}}_i^k = \mathbf{z}_i^k$ for $i \in S^k$ and $\hat{\mathbf{z}}_i^k = \mathbf{x}_i^k$ for $i \notin S^k$

(S.4) : Set $\mathbf{x}^{k+1} \triangleq \mathbf{x}^k + \gamma^k (\hat{\mathbf{z}}^k - \mathbf{x}^k)$;

(S.5) : $k \leftarrow k + 1$, and go to (S.1).

Theorem 1: Let $\{\mathbf{x}^k\}$ be the sequence generated by Algorithm 1, under A1-A6. Suppose that $\{\gamma^k\}$ and $\{\varepsilon_i^k\}$ satisfy the following conditions: i) $\gamma^k \in (0, 1]$; ii) $\gamma^k \rightarrow 0$; iii) $\sum_k \gamma^k = +\infty$; iv) $\sum_k (\gamma^k)^2 < +\infty$; and v) $\varepsilon_i^k \leq \gamma^k \alpha_1 \min\{\alpha_2, 1/\|\nabla_{\mathbf{x}_i} F(\mathbf{x}^k)\|\}$ for all $i \in \mathcal{N}$ and some nonnegative constants α_1 and α_2 . Additionally, if inexact solutions are used in Step 2, i.e., $\varepsilon_i^k > 0$ for some i and infinite k , then assume also that G is globally Lipschitz on X . Then, either Algorithm 1 converges in a finite number of iterations to a stationary solution of (1) or every limit point of $\{\mathbf{x}^k\}$ (at least one such points exists) is a stationary solution of (1).

Proof: See Appendix B. ■

The proposed algorithm is extremely flexible. We can always choose $S^k = \mathcal{N}$ resulting in the simultaneous update of all the (block) variables (full Jacobi scheme); or, at the other extreme, one can update a single (block) variable per time, thus obtaining a Gauss-Southwell kind of method. More classical cyclic Gauss-Seidel methods can also be derived and are discussed in the next subsection. One can also compute inexact solutions (Step 2) while preserving convergence, provided that the error term ε_i^k and the step-size γ^k ’s are chosen according to Theorem 1; some practical choices for these parameters are discussed in Section IV. We emphasize that the Lipschitzianity of G is required only if $\hat{\mathbf{x}}(\mathbf{x}^k, \boldsymbol{\tau})$ is not computed exactly for *infinite* iterations. At any rate this Lipschitz conditions is

¹We recall that a stationary solution \mathbf{x}^* of (1) is a points for which a subgradient $\xi \in \partial G(\mathbf{x}^*)$ exists such that $(\nabla F(\mathbf{x}^*) + \xi)^T (\mathbf{y} - \mathbf{x}^*) \geq 0$ for all $\mathbf{y} \in X$. Of course, if F is convex, stationary points coincide with global minimizers.

automatically satisfied if G is a norm (and therefore in LASSO and group LASSO problems for example) or if X is bounded.

As a final remark, note that versions of Algorithm 1 where all (or most of) the variables are updated at each iteration are particularly amenable to implementation in *distributed* environments (e.g., multi-user communications systems, ad-hoc networks, etc.). In fact, in this case, not only the calculation of the inexact solutions z_i^k can be carried out in parallel, but the information that “the i -th subproblem” has to exchange with the “other subproblem” in order to compute the next iteration is very limited. A full appreciation of the potentialities of our approach in distributed settings depends however on the specific application under consideration and is beyond the scope of this paper. We refer the reader to [21] for some examples, even if in less general settings.

A. A Gauss-Jacobi algorithm

Algorithm 1 and its convergence theory cover fully parallel Jacobi as well as Gauss-Southwell-type methods, and many of their variants. In this section we show that Algorithm 1 can also incorporate *hybrid parallel-sequential* (Jacobi–Gauss-Seidel) schemes wherein block of variables are updated *simultaneously* by *sequentially* computing entries per block. This procedure seems particularly well suited to parallel optimization on multi-core/processor architectures.

Suppose that we have P processors that can be used in parallel and we want to exploit them to solve Problem (1) (P will denote both the number of processors and the set $\{1, 2, \dots, P\}$). We “assign” to each processor p the variables I_p ; therefore I_1, \dots, I_P is a partition of I . We denote by $\mathbf{x}_p \triangleq (\mathbf{x}_{pi})_{i \in I_p}$ the vector of (block) variables \mathbf{x}_{pi} assigned to processor p , with $i \in I_p$; and \mathbf{x}_{-p} is the vector of remaining variables, i.e., the vector of those assigned to all processors except the p -th one. Finally, given $i \in I_p$, we partition \mathbf{x}_p as $\mathbf{x}_p = (\mathbf{x}_{pi<}, \mathbf{x}_{pi\geq})$, where $\mathbf{x}_{pi<}$ is the vector containing all variables in I_p that come before i (in the order assumed in I_p), while $\mathbf{x}_{pi\geq}$ are the remaining variables. Thus we will write, with a slight abuse of notation $\mathbf{x} = (\mathbf{x}_{pi<}, \mathbf{x}_{pi\geq}, \mathbf{x}_{-p})$.

Once the optimization variables have been assigned to the processors, one could in principle apply the inexact Jacobi Algorithm 1. In this scheme each processor p would compute *sequentially*, at each iteration k and for every (block) variable \mathbf{x}_{pi} , a suitable \mathbf{z}_{pi}^k by keeping all variables but \mathbf{x}_{pi} fixed to $(\mathbf{x}_{pj}^k)_{i \neq j \in I_p}$ and \mathbf{x}_{-p}^k . But since we are solving the problems for each group of variables assigned to a processor sequentially, this seems a waste of resources; it is instead much more efficient to use, within each processor, a Gauss-Seidel scheme, whereby the *current* calculated iterates are used in all subsequent calculations. Our Gauss-Jacobi method formally described in Algorithm 2 implements exactly this idea; its convergence properties are given in Theorem 2.

Theorem 2: Let $\{\mathbf{x}^k\}_{k=1}^\infty$ be the sequence generated by Algorithm 2, under the setting of Theorem 1. Then, either Algorithm 2 converges in a finite number of iterations to a stationary solution of (1) or every limit point of the sequence $\{\mathbf{x}^k\}_{k=1}^\infty$ (at least one such points exists) is a stationary solution of (1).

Proof: See Appendix C. ■

Algorithm 2: Inexact Gauss-Jacobi Algorithm

Data : $\{\varepsilon_{pi}^k\}$ for $p \in P$ and $i \in I_p$, $\tau \geq 0$, $\{\gamma^k\} > 0$, $\mathbf{x}^0 \in \mathcal{K}$. Set $k = 0$.

(S.1) : If \mathbf{x}^k satisfies a termination criterion: STOP;

(S.2) : For all $p \in P$ do (in parallel),

For all $i \in I_p$ do (sequentially)

a) Find \mathbf{z}_{pi}^k s.t.

$$\|\mathbf{z}_{pi}^k - \hat{\mathbf{x}}_{pi}((\mathbf{x}_{pi<}^{k+1}, \mathbf{x}_{pi\geq}^k, \mathbf{x}_{-p}^k), \tau)\| \leq \varepsilon_{pi}^k;$$

b) Set $\mathbf{x}_{pi}^{k+1} \triangleq \mathbf{x}_{pi}^k + \gamma^k (\mathbf{z}_{pi}^k - \mathbf{x}_{pi}^k)$

(S.3) : $k \leftarrow k + 1$, and go to (S.1).

Although the proof of Theorem 2 is relegated to the appendix, it is interesting to point out that the gist of the proof is to show that Algorithm 2 is nothing else but an instance of Algorithm 1 with errors.

By updating all variables at each iteration, Algorithm 2 has the advantage that neither the error bounds E_i nor the exact solutions $\hat{\mathbf{x}}_{pi}$ need to be computed, in order to decide which variables should be updated. Furthermore it is rather intuitive that the use of the “latest available information” should reduce the number of overall iterations needed to converge with respect to Algorithm 1 (assuming in the latter algorithm that all variables are updated at each iteration). However this advantages should be contrasted with the following two facts: i) updating all variables at each iteration might not always be the best (or a feasible) choice; and ii) in many practical instances of Problem (1), using the latest information as dictated by Algorithm 2 may require extra calculations (e.g., to compute function information, as the gradients) and communication overhead. These aspects are discussed on specific examples in Section VI.

As a final remark, note that Algorithm 2 contains as special case the classical cyclical Gauss-Seidel scheme (a fact that was less obvious to deduce directly from Algorithm 1); it is sufficient to set $P = 1$ (corresponding to using only one processor): the single processor updates all the (scalar) variables sequentially while using the new values of those that have already been updated.

IV. EXAMPLES AND SPECIAL CASES

Algorithms 1 and 2 are very general and encompass a gamut of novel algorithms, each corresponding to various forms of the approximant P_i , the error bound function E_i , the step-size sequence γ^k , the block partition, etc. These choices lead to algorithms that can be very different from each other, but *all converging under the same conditions*. These degrees of freedom offer a lot of flexibility to control iteration complexity, communication overhead, and convergence speed. In this section we outline several effective choices for the design parameters along with some illustrative examples of specific algorithms resulting from a proper combination of these choices.

On the choice of the step-size γ^k . An example of step-size rule satisfying conditions i)-iv) in Theorem 1 is: given $0 <$

$\gamma^0 \leq 1$, let

$$\gamma^k = \gamma^{k-1} (1 - \theta \gamma^{k-1}), \quad k = 1, \dots, \quad (5)$$

where $\theta \in (0, 1)$ is a given constant. Notice that while this rule may still require some tuning for optimal behavior, it is quite reliable, since in general we are not using a (sub)gradient direction, so that many of the well-known practical drawbacks associated with a (sub)gradient method with diminishing step-size are mitigated in our setting. Furthermore, this choice of step-size does not require any form of centralized coordination, which is a favorable feature in a parallel environment. Numerical results in Section VI show the effectiveness of (the customization of) (5) on specific problems.

We remark that it is possible to prove convergence of Algorithm 1 also using other step-size rules, such as a standard Armijo-like line-search procedure or a (suitably small) constant step-size. We omit the discussion of these options because the former is not in line with our parallel approach while the latter is numerically less efficient.

On the choice of the error bound function $E_i(\mathbf{x})$.

- As we mentioned, the most obvious choice is to take $E_i(\mathbf{x}) = \|\hat{\mathbf{x}}_i(\mathbf{x}^k, \tau_i) - \mathbf{x}_i^k\|$. This is a valuable choice if the computation of $\hat{\mathbf{x}}_i(\mathbf{x}^k, \tau_i)$ can be easily accomplished. For instance, in the LASSO problem with $\mathcal{N} = \{1, \dots, n\}$ (i.e., when each block reduces to a scalar variable), it is well-known that $\hat{\mathbf{x}}_i(\mathbf{x}^k, \tau_i)$ can be computed in closed form using the soft-thresholding operator [12].
- In situations where the computation of $\|\hat{\mathbf{x}}_i(\mathbf{x}^k, \tau_i) - \mathbf{x}_i^k\|$ is not possible or advisable, we can resort to estimates. Assume momentarily that $G \equiv 0$. Then it is known [27, Proposition 6.3.1] under our assumptions that $\|\Pi_{X_i}(\mathbf{x}_i^k - \nabla_{\mathbf{x}_i} F(\mathbf{x}^k)) - \mathbf{x}_i^k\|$ is an error bound for the minimization problem in (3) and therefore satisfies (4), where $\Pi_{X_i}(\mathbf{y})$ denotes the Euclidean projection of \mathbf{y} onto the closed and convex set X_i . In this situation we can choose $E_i(\mathbf{x}^k) = \|\Pi_{X_i}(\mathbf{x}_i^k - \nabla_{\mathbf{x}_i} F(\mathbf{x}^k)) - \mathbf{x}_i^k\|$. If $G(\mathbf{x}) \neq 0$ things become more complex. In most cases of practical interest, adequate error bounds can be derived from [11, Lemma 7].
- It is interesting to note that the computation of E_i is only needed if a partial update of the (block) variables is performed. However, an option that is always feasible is to take $S^k = \mathcal{N}$ at each iteration, i.e., update all (block) variables at each iteration. With this choice we can dispense with the computation of E_i altogether.

On the choice of the approximant $P_i(\mathbf{x}_i; \mathbf{x})$.

- The most obvious choice for P_i is the linearization of F at \mathbf{x}^k with respect to \mathbf{x}_i :

$$P_i(\mathbf{x}_i; \mathbf{x}^k) = F(\mathbf{x}^k) + \nabla_{\mathbf{x}_i} F(\mathbf{x}^k)^T (\mathbf{x}_i - \mathbf{x}_i^k).$$

With this choice, and taking for simplicity $\mathbf{Q}_i(\mathbf{x}^k) = \mathbf{I}$,

$$\hat{\mathbf{x}}_i(\mathbf{x}^k, \tau_i) = \underset{\mathbf{x}_i \in X_i}{\operatorname{argmin}} \left\{ F(\mathbf{x}^k) + \nabla_{\mathbf{x}_i} F(\mathbf{x}^k)^T (\mathbf{x}_i - \mathbf{x}_i^k) + \frac{\tau_i}{2} \|\mathbf{x}_i - \mathbf{x}_i^k\|^2 + g_i(\mathbf{x}_i) \right\}. \quad (6)$$

This is essentially the way a new iteration is computed in most *sequential* (block-)CDMs for the solution of (group) LASSO problems and its generalizations. Note that contrary to most

existing schemes, our algorithm is *parallel*.

- At another extreme we could just take $P_i(\mathbf{x}_i; \mathbf{x}^k) = F(\mathbf{x}_i, \mathbf{x}_{-i}^k)$. Of course, to have P1 satisfied (cf. Section III), we must assume that $F(\mathbf{x}_i, \mathbf{x}_{-i}^k)$ is convex. With this choice, and setting for simplicity $\mathbf{Q}_i(\mathbf{x}^k) = \mathbf{I}$, we have

$$\hat{\mathbf{x}}_i(\mathbf{x}^k, \tau_i) \triangleq \underset{\mathbf{x}_i \in X_i}{\operatorname{argmin}} \left\{ F(\mathbf{x}_i, \mathbf{x}_{-i}^k) + \frac{\tau_i}{2} \|\mathbf{x}_i - \mathbf{x}_i^k\|^2 + g_i(\mathbf{x}_i) \right\}, \quad (7)$$

thus giving rise to a parallel nonlinear Jacobi type method for the constrained minimization of $V(\mathbf{x})$.

- Between the two “extreme” solutions proposed above, one can consider “intermediate” choices. For example, If $F(\mathbf{x}_i, \mathbf{x}_{-i}^k)$ is convex, we can take $P_i(\mathbf{x}_i; \mathbf{x}^k)$ as a second order approximation of $F(\mathbf{x}_i, \mathbf{x}_{-i}^k)$, i.e.,

$$P_i(\mathbf{x}_i; \mathbf{x}^k) = F(\mathbf{x}^k) + \nabla_{\mathbf{x}_i} F(\mathbf{x}^k)^T (\mathbf{x}_i - \mathbf{x}_i^k) + \frac{1}{2} (\mathbf{x}_i - \mathbf{x}_i^k)^T \nabla_{\mathbf{x}_i \mathbf{x}_i}^2 F(\mathbf{x}^k) (\mathbf{x}_i - \mathbf{x}_i^k). \quad (8)$$

When $g_i(\mathbf{x}_i) \equiv 0$, this essentially corresponds to taking a Newton step in minimizing the “reduced” problem $\min_{\mathbf{x}_i \in X_i} F(\mathbf{x}_i, \mathbf{x}_{-i}^k)$, resulting in

$$\hat{\mathbf{x}}_i(\mathbf{x}^k, \tau_i) = \underset{\mathbf{x}_i \in X_i}{\operatorname{argmin}} \left\{ F(\mathbf{x}^k) + \nabla_{\mathbf{x}_i} F(\mathbf{x}^k)^T (\mathbf{x}_i - \mathbf{x}_i^k) + \frac{1}{2} (\mathbf{x}_i - \mathbf{x}_i^k)^T \nabla_{\mathbf{x}_i \mathbf{x}_i}^2 F(\mathbf{x}^k) (\mathbf{x}_i - \mathbf{x}_i^k) + \frac{\tau_i}{2} \|\mathbf{x}_i - \mathbf{x}_i^k\|^2 + g_i(\mathbf{x}_i) \right\}. \quad (9)$$

- Another “intermediate” choice, relying on a specific structure of the objective function that has important applications is the following. Suppose that F is a sum-utility function, i.e.,

$$F(\mathbf{x}) = \sum_{j \in J} f_j(\mathbf{x}_i, \mathbf{x}_{-i}),$$

for some finite set J . Assume now that for every $j \in S_i \subseteq J$, the functions $f_j(\bullet, \mathbf{x}_{-i})$ is convex. Then we may set

$$P_i(\mathbf{x}_i; \mathbf{x}^k) = \sum_{j \in S_i} f_j(\mathbf{x}_i, \mathbf{x}_{-i}^k) + \sum_{j \notin S_i} \nabla f_j(\mathbf{x}_i, \mathbf{x}_{-i}^k)^T (\mathbf{x}_i - \mathbf{x}_i^k)$$

thus preserving, for each i , the favorable convex part of F with respect to \mathbf{x}_i while linearizing the nonconvex parts. This is the approach adopted in [21] in the design of multi-users systems, to which we refer for applications in signal processing and communications.

The framework described in Algorithm 1 can give rise to very different schemes, according to the choices one makes for the many variable features it contains, some of which have been detailed above. Because of space limitation, we cannot discuss here all possibilities. We provide next just a few instances of possible algorithms that fall in our framework.

Example #1—(Proximal) Jacobi algorithms for convex functions. Consider the simplest problem falling in our setting: the unconstrained minimization of a continuously differentiable convex function, i.e., assume that F is convex, $G \equiv 0$, and $X = \mathbb{R}^n$. Although this is possibly the best studied problem in nonlinear optimization, classical parallel methods for this problem [26, Sec. 3.2.4] require very strong contraction conditions. In our framework we can take $P_i(\mathbf{x}_i; \mathbf{x}^k) = F(\mathbf{x}_i, \mathbf{x}_{-i}^k)$, resulting in a parallel Jacobi-type method which does not need any additional assumptions. Furthermore our

theory shows that we can even dispense with the convexity assumption and still get convergence of a Jacobi-type method to a stationary point. If in addition we take $S^k = \mathcal{N}$, we obtain the class of methods studied in [21], [28]–[30].

Example #2–Parallel coordinate descent methods for LASSO. Consider the LASSO problem, i.e., Problem (1) with $F(\mathbf{x}) = \|\mathbf{Ax} - \mathbf{b}\|^2$, $G(\mathbf{x}) = c\|\mathbf{x}\|_1$, and $X = \mathbb{R}^n$. Probably, to date, the most successful class of methods for this problem is that of CDMs, whereby at each iteration a *single* variable is updated using (6). We can easily obtain a parallel version for this method by taking $n_i = 1$, $S^k = \mathcal{N}$ and still using (6). Alternatively, instead of linearizing $F(\mathbf{x})$, we can better exploit the structure of $F(\mathbf{x})$ and use (7). In fact, it is well known that in LASSO problems subproblem (7) can be solved analytically. We can easily consider similar methods for the group LASSO problem as well (just take $n_i > 1$).

Example #3–Parallel coordinate descent methods for Logistic Regression. Consider the Logistic Regression problem, i.e., Problem (1) with $F(\mathbf{x}) = \sum_{j=1}^m \log(1 + e^{-a_j \mathbf{y}_j^T \mathbf{x}})$, $G(\mathbf{x}) = c\|\mathbf{x}\|_1$, and $X = \mathbb{R}^n$, where $\mathbf{y}_i \in \mathbb{R}^n$, $a_i \in \{-1, 1\}$, and $c \in \mathbb{R}_{++}$ are given constants. Since $F(\mathbf{x}_i, \mathbf{x}_{-i}^k)$ is convex, we can take $P_i(\mathbf{x}_i; \mathbf{x}^k) = F(\mathbf{x}^k) + \nabla_{\mathbf{x}_i} F(\mathbf{x}^k)^T (\mathbf{x}_i - \mathbf{x}_i^k) + \frac{1}{2}(\mathbf{x}_i - \mathbf{x}_i^k)^T \nabla_{\mathbf{x}_i \mathbf{x}_i}^2 F(\mathbf{x}^k) (\mathbf{x}_i - \mathbf{x}_i^k)$ and thus obtaining a fully distributed and parallel CDM that uses a second order approximation of the smooth function F . Moreover by taking $n_i = 1$ and using a soft-thresholding operator, each $\hat{\mathbf{x}}_i$ can be computed in closed form.

V. RELATED WORKS

The proposed algorithmic framework draws on Successive Convex Approximation (SCA) paradigms that have a long history in the optimization literature. Nevertheless, our algorithms and their convergence conditions (cf. Theorems 1 and 2) unify and extend current parallel and sequential SCA methods in several directions, as outlined next.

(Partially) Parallel Deterministic Methods: The roots of parallel deterministic SCA schemes (wherein *all* the variables are updated simultaneously) can be traced back at least to the work of Cohen on the so-called auxiliary principle [28], [29] and its related developments, see e.g. [9]–[16], [21], [30]–[32]. Roughly speaking these works can be divided in two groups, namely: solution methods for *convex* objective functions [9], [12], [14]–[16], [28], [29] and *nonconvex* ones [10], [11], [13], [21], [30]–[32]. All methods in the former group (and [10], [11], [13], [31], [32]) are (proximal) gradient schemes; they thus share the classical drawbacks of gradient-like schemes; moreover, by replacing the convex function F with its first order approximation, they do not take any advantage of the structure of F , a fact that instead has been shown to enhance convergence speed [21]. Comparing with the second group of works [10], [11], [13], [21], [30]–[32], our algorithmic framework improves on their convergence properties while adding more flexibility in the selection of how many variables to update at each iteration. For instance, with the exception of [11], all the aforementioned works do not allow parallel updates of only a *subset* of all variables, a fact that instead can dramatically improve the convergence speed of the algorithm,

as we show in Section VI. Moreover, with the exception of [30], they all require an Armijo-type line-search, which makes them not appealing for a (parallel) distributed implementation. A scheme in [30] is actually based on diminishing step-size-rules, but its convergence properties are quite weak: not all the limit points of the sequence generated by this scheme are guaranteed to be stationary solutions of (1).

Our framework instead i) deals with *nonconvex* (nonsmooth) problems; ii) allows one to use a much varied array of approximations for F and also inexact solutions of the subproblems; iii) is fully parallelizable and distributable (it does not rely on any line-search); and iv) leads to the *first distributed convergent* schemes based on very general (possibly) *partial* updating rules of the optimization variables. In fact, among deterministic schemes, we are aware of only three algorithms [11], [14], [15] performing at each iteration a parallel update of only a *subset* of all the variables. These algorithms however are gradient-like schemes, and do not allow inexact solutions of the subproblems (in some large-scale problems the cost of computing the exact solution of all the subproblems can be prohibitive). In addition, [11] requires an Armijo-type line-search whereas [14] and [15] are applicable only to *convex* objective functions and are not *fully* parallel. In fact, convergence conditions therein impose a constraint on the maximum number of variables that can be simultaneously updated (linked to the spectral radius of some matrices), a constraint that in many large scale problems is likely not satisfied.

Sequential Methods: Our framework contains as special cases also sequential updates; it is then interesting to compare our results to sequential schemes too. Given the vast literature on the subject, we consider here only the most recent and general work [17]. In [17] the authors consider the minimization of a possibly nonsmooth function by Gauss-Seidel methods whereby, at each iteration, a *single* block of variables is updated by minimizing a *global upper* convex approximation of the function. However, finding such an approximation is generally not an easy task, if not impossible. To cope with this issue, the authors also proposed a variant of their scheme that does not need this requirement but uses an Armijo-type line-search, which however makes the scheme not suitable for a parallel/distributed implementation. Contrary to [17], in our framework conditions on the approximation function (cf. P1-P3) are trivial to be satisfied (in particular, P need not be an upper bound of F), enlarging significantly the class of utility functions V which the proposed solution method is applicable to. Furthermore, our framework gives rise to parallel and distributed methods (no line search is used) wherein all variables can be updated rather independently at the same time.

VI. NUMERICAL RESULTS

In this section we provide some numerical results providing a solid evidence of the viability of our approach; they clearly show that our algorithmic framework leads to practical methods that exploit well parallelism and compare favourably to existing schemes, both parallel and sequential. The tests were carried out on LASSO and Logistic Regression problems, two of the most studied instances of Problem (1).

All codes have been written in C++ and use the Message Passing Interface for parallel operations. All algebra is performed by using the GNU Scientific Library (GSL). The algorithms were tested on the General Compute Cluster of the Center for Computational Research at the State University of New York at Buffalo. In particular for our experiments we used a partition composed of 372 DELL 12x2.40GHz Intel Xeon E5645 Processor computer nodes with 48 Gb of main memory and QDR InfiniBand 40Gb/s network card. In our experiments distributed algorithms ran on 20 parallel processes (that is we used 2 nodes with 10 cores each one), while sequential algorithms ran on a single process (using thus one single core).

A. LASSO problem

We implemented the instance of Algorithm 1 that we described in Example # 2 in the previous section, using the approximating function P_i as in (7). Note that in the case of LASSO problems $\hat{\mathbf{x}}_i(\mathbf{x}^k, \tau_i)$, the unique solution (7), can be easily computed in closed form using the soft-thresholding operator, see e.g. [12].

Tuning of Algorithm 1: The free parameters of our algorithm are chosen as follows. The proximal gains τ_i are initially all set to $\tau_i = \text{tr}(\mathbf{A}^T \mathbf{A})/2n$, where n is the total number of variables. This initial value, which is half of the mean of the eigenvalues of $\nabla^2 F$, has been observed to be very effective in all our numerical tests. Choosing an appropriate value of τ_i at each iteration is crucial. Note that in the description of our algorithmic framework we considered fixed values of τ_i , but it is clear that varying them a finite number of times does not affect in any way the theoretical convergence properties of the algorithms. On the other hand, we found that an appropriate update of τ_i in early iterations can enhance considerably the performance of the algorithm. Some preliminary experiments showed that an effective option is to choose τ_i “large enough” to force a decrease in the objective function value, but not “too large” to slow down progress towards optimality. We found that the following heuristic works well in practice: (i) all τ_i are doubled if at a certain iteration the objective function does not decrease; and (ii) they are all halved if the objective function decreases for ten consecutive iterations or the relative error on the objective function $\text{re}(\mathbf{x})$ is sufficiently small, specifically if

$$\text{re}(\mathbf{x}) \triangleq \frac{V(\mathbf{x}) - V^*}{V^*} \leq 10^{-2}, \quad (10)$$

where V^* is the optimal value of the objective function V (in our experiments on LASSO V^* is known, see below). In order to avoid increments in the objective function, whenever all τ_i are doubled, the associated iteration is discarded, and in Step 4 of Algorithm 1 it is set $\mathbf{x}^{k+1} = \mathbf{x}^k$. In any case we limited the number of possible updates of the values of τ_i to 100.

The step-size γ^k is updated according to the following rule:

$$\gamma^k = \gamma^{k-1} \left(1 - \min \left\{ 1, \frac{10^{-4}}{\text{re}(\mathbf{x}^{k-1})} \right\} \theta \gamma^{k-1} \right), \quad k = 1, \dots, \quad (11)$$

with $\gamma^0 = 0.9$ and $\theta = 1e - 7$. The above diminishing rule is based on (5) while guaranteeing that γ^k does not become too close to zero before the relative error is sufficiently small. Note that since τ_i are changed only a finite number of times

and the step-size γ^k decreases, the conditions of Theorem 1 are all satisfied.

Finally the error bound function is chosen as $E_i(\mathbf{x}^k) = \|\hat{\mathbf{x}}_i(\mathbf{x}^k, \tau_i) - \mathbf{x}_i^k\|$, and S^k in Step 3 of the algorithm is set to

$$S^k = \{i : E_i(\mathbf{x}^k) \geq \sigma M^k\}.$$

In our tests we consider two options for σ , namely: i) $\sigma = 0$, which leads to a *fully parallel* scheme wherein at each iteration *all* variables are updated; and ii) $\sigma = 0.5$, which corresponds to updating only a subset of all the variables at each iteration. Note that for both choices of σ , the resulting set S^k satisfies the requirement in Step 3 of Algorithm 1; indeed, S^k always contains the index i corresponding to the largest $E_i(\mathbf{x}^k)$. Recall also that, as we already mentioned, the computation of each $\hat{\mathbf{x}}_i(\mathbf{x}^k, \tau_i)$ for the LASSO problem is in closed form and thus inexpensive.

We termed the above instance of our Algorithm 1 FLEXible parallel Algorithm (FLEXA); in the sequel we will refer to the two versions of FLEXA as FLEXA $\sigma = 0$ and FLEXA $\sigma = 0.5$.

Algorithms in the literature: We compared our versions of FLEXA with the most common distributed and sequential algorithms proposed in the literature to solve the LASSO problem. More specifically, we consider the following schemes.

- **FISTA:** The Fast Iterative Shrinkage-Thresholding Algorithm (FISTA) proposed in [12] is a first order method and can be regarded as the benchmark algorithm for LASSO problems. By taking advantages of the separability of the terms in the objective function V , this method can be easily parallelized and thus implemented on a parallel architecture. FISTA requires the preliminary computation of the Lipschitz constant L_F of ∇F ; in our experiments we performed this computation using a distributed version of the power method that computes $\|\mathbf{A}\|_2^2$ (see, e.g., [33]).

- **SpaRSA:** This is the first order method proposed in [13]; it is a popular spectral projected gradient method that uses a spectral step length together with a nonmonotone line search to enhance convergence. Also this method can be easily parallelized, which is the version that implemented in our tests. In all the experiments we set the parameters of SpaRSA as in [13]: $M = 5$, $\sigma = 0.01$, $\alpha_{\max} = 1e30$, and $\alpha_{\min} = 1e - 30$.

- **GRock:** This is a parallel algorithm proposed in [15] that seems to perform extremely well on sparse LASSO problems. We actually tested two instances of GRock, namely: i) one wherein only one variable is updated at each iteration; and ii) a second instance where the number of variables simultaneously updated is equal to the number of the parallel processors (in our experiments we used 20 processors). It is important to remark that the theoretical convergence properties of GRock are in jeopardy as the number of variables updated in parallel increases; roughly speaking, GRock is guaranteed to converge if the columns of the data matrix \mathbf{A} in the LASSO problem are “almost” orthogonal, a feature enjoyed by most of our test problems, but that is not satisfied in many applications.

- **ADMM:** This is a classical Alternating Method of Multipliers (ADMM) in the form used in [34]. Applied to LASSO problems, this instance leads to a sequential scheme where

only one variable per time can be updated (in closed form). Note that in principle ADMM can be parallelized, but it is well known that it does not scale well with the number of the processors; therefore in our tests we have not implemented the parallel version.

- **GS:** This is a classical sequential Gauss-Seidel scheme [26] computing \hat{x}_i with $n_i = 1$, and then updating all x_i in a sequential fashion (and using unitary step-size).

In all the parallel algorithms we implemented (FLEXA, FISTA, SpaRSA and GRock), the data matrix \mathbf{A} of the LASSO problem is stored in a column block distributed manner $\mathbf{A} = [\mathbf{A}_1 \mathbf{A}_2 \cdots \mathbf{A}_P]$, where P is the number of parallel processors. Thus the computation of each product $\mathbf{A}\mathbf{x}$ (which is required to evaluate ∇F) and the norm $\|\mathbf{x}\|_1$ (that is G) is divided into the parallel jobs of computing $\mathbf{A}_i\mathbf{x}_i$ and $\|\mathbf{x}_i\|_1$, followed by a reduce operation. Columns of \mathbf{A} were equally distributed among the processes.

Numerical Tests: We generated six groups of LASSO problems using the random generation technique proposed by Nesterov [10]; this method permits to control the sparsity of the solution. For the first five groups, we considered problems with 10,000 variables and matrix \mathbf{A} having 9,000 rows. The five groups differ from the degree of sparsity of the solution; more specifically the percentage of non zeros in the solution is 1%, 10%, 20%, 30%, and 40%, respectively. The last group is formed by instances with 100,000 variables and 5000 rows for \mathbf{A} , and solutions having 1% of non zero variables. In all experiments and for all the algorithms, the initial point was set to the zero vector.

Results of our experiments for each of the 10,000 variables groups are reported in Fig. 1, where we plot the relative error as defined in (10) versus the CPU time; all the curves are averaged over ten independent random realizations. Note that the CPU time includes communication times (for distributed algorithms) and the initial time needed by the methods to perform all pre-iterations computations (this explains why the curves associated with FISTA start after the others; in fact FISTA requires some nontrivial initializations based on the computation of $\|\mathbf{A}\|_2^2$).

Results of our experiments for the LASSO instance with 100,000 variables are reported in Fig. 2. The curves are averaged over three random realizations. Note that we have not included the curves for sequential algorithms (ADMM and GS) on this group of big problems, since we could not use the same nodes used to run all the other algorithms, due to memory limitations. However, we tested ADMM and GS on these big problems on different high-memory nodes; the obtained results (not reported here) showed that, as the dimensions of the problem increase, sequential methods perform poorly in comparison with parallel methods; therefore we excluded ADMM and GS in the tests for the LASSO instance with 100,000 variables.

Given Fig. 1 and 2, the following comments are in order. On all the tested problems, FLEXA with $\sigma = 0.5$ *outperforms in a consistent manner all other implemented algorithms*. Results for FLEXA with $\sigma = 0$ are quite similar to those with $\sigma = 0.5$ on the 10,000 variables problems. However on larger problems FLEXA $\sigma = 0$ (i.e., the version wherein all variables

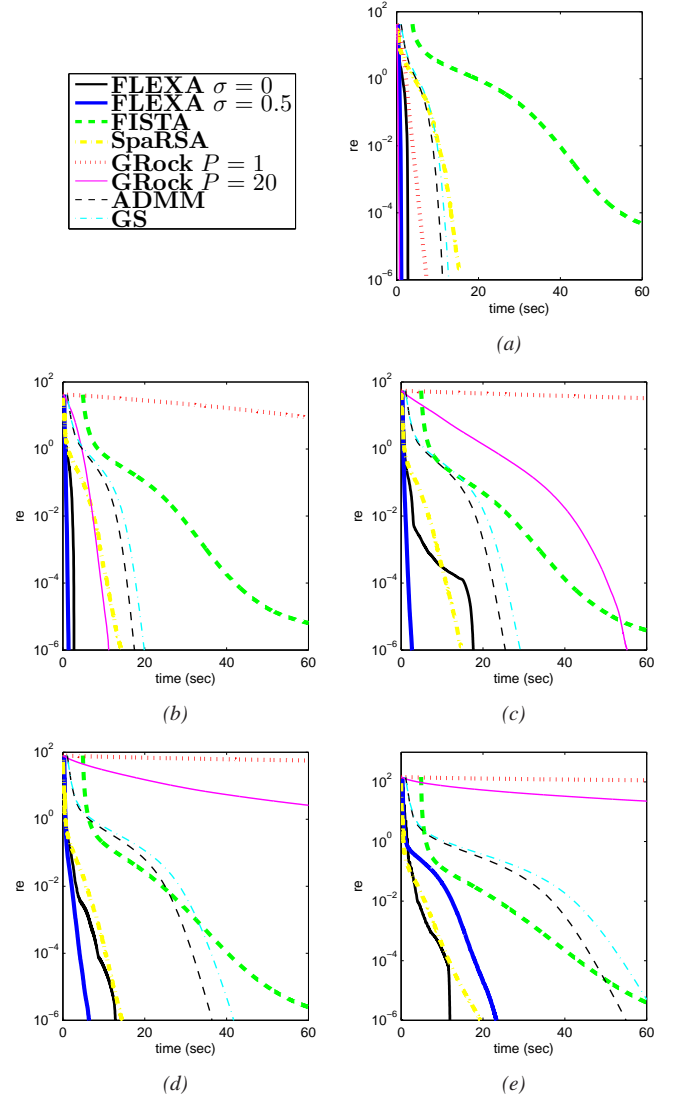


Fig. 1: Relative error vs. time (in seconds) for Lasso with 10,000 variables: (a) 1% non zeros - (b) 10% non zeros - (c) 20% non zeros - (d) 30% non zeros - (e) 40% non zeros

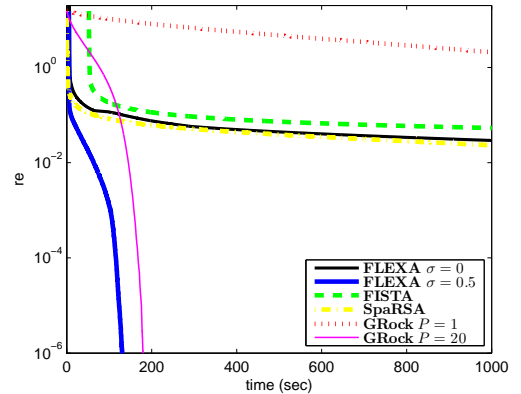


Fig. 2: Relative error vs. time (in seconds) for Lasso with 100,000 variables

are updated at each iteration) seems ineffective. This result might seem surprising at first sight: why, once all the optimal solutions $\hat{x}_i(\mathbf{x}^k, \tau_i)$ are computed, is it more convenient not to use all of them but update instead only a subset of variables? We briefly discuss this complex issue next.

Remark 3 (On the partial updates): It can be shown that

Algorithm 1 has the remarkable capability to *identify those variables that will be zero at a solution*; because of lack of space, we do not provide here the proof of this statement but only an informal description. Roughly speaking, it can be shown that, for k large enough, those variables that are zero in $\hat{\mathbf{x}}(\mathbf{x}^k, \tau)$ will be zero also in a limiting solution $\bar{\mathbf{x}}$. Therefore, suppose that k is large enough so that this identification property already takes place (we will say that “we are in the identification phase”) and consider an index i such that $\bar{x}_i = 0$. Then, if x_i^k is zero, it is clear, by Steps 3 and 4, that $x_i^{k'}$ will be zero for all indices $k' > k$, independently of whether i belongs to S^k or not. In other words, if a variable that is zero at the solution is already zero when the algorithm enters the identification phase, *that variable will be zero in all subsequent iterations*; this fact, intuitively, should enhance the convergence speed of the algorithm. Conversely, if when we enter the identification phase x_i^k is not zero, the algorithm will have to bring it back to zero iteratively. It should then be clear why updating only variables that we have “strong” reason to believe will be non zero at a solution is a better strategy than updating them all. Of course, there may be a problem dependence and the best value of σ can vary from problem to problem. But we believe that the explanation outlined above gives firm theoretical ground to the idea that it might be wise to “waste” some calculations and perform only a partial update of the variables. \square

Referring to sequential methods (ADMM and GS), they behave strikingly well on the 10,000 variables problems, if one keeps in mind that they only use one process. However, as already observed, they cannot compete with parallel methods on larger problems. FISTA is capable to approach relatively fast low accuracy solutions, but has difficulties in reaching high accuracy. The version of GRock with $P = 20$ is the closest match to FLEXA, but only when the problems are very sparse. This is consistent with the fact that its convergence properties are at stake when the problems are quite dense. Furthermore, it should be clear that if the problem is very large, updating only 20 variables at each iteration, as GRock does, could slow down the convergence, especially when the optimal solution is not very sparse. From this point of view, the strategy used by FLEXA $\sigma = 0.5$ seems to strike a good balance between not updating variables that are probably zero at the optimum and nevertheless update a sizeable amount of variables when needed in order to enhance convergence. Finally, SpaRSA seems to be very insensitive to the degree of sparsity of the solution; it is comparable to our FLEXA on 10,000 variables problems, but is much less effective on very large-scale problems. In conclusion, Fig. 1 and Fig. 2 show that while there is no algorithm in the literature performing equally well on all the simulated (large and very large-scale) problems, the proposed FLEXA is consistently the “winner”.

B. Logistic regression problems

The logistic regression problem is described in Example #3 (cf. Section III). For such a problem, we implemented the instance of Algorithm 1 described in the same example. More specifically, the algorithm is essentially the same described for LASSO, but with the following differences:

Data set	m	n	c
gisette (scaled)	6000	5000	1/1500
colon-cancer	62	2000	0.01
leukemia	38	7129	0.01

TABLE I: Test problems for logistic regression tests

- The approximant P_i is chosen as the second order approximation of the original function F ;
- The initial τ_i are set to $\text{tr}(\mathbf{Y}^T \mathbf{Y})/2n$ for all i , where n is the total number of variables and $\mathbf{Y} = [\mathbf{y}_1 \mathbf{y}_2 \cdots \mathbf{y}_m]^T$.
- Since the optimal value V^* is not known for the logistic regression problem, we no longer use $\text{re}(\mathbf{x})$ as merit function but $\|Z(\mathbf{x})\|_\infty$, with

$$Z(\mathbf{x}) = \nabla F(\mathbf{x}) - \Pi_{[-c, c]^n}(\nabla F(\mathbf{x}) - \mathbf{x}).$$

Here the projection $\Pi_{[-c, c]^n}(\mathbf{z})$ can be efficiently computed; it acts component-wise on \mathbf{z} , since $[-c, c]^n = [-c, c] \times \cdots \times [-c, c]$. Note that $Z(\mathbf{x})$ is a valid optimality measure function; indeed, $Z(\mathbf{x}) = 0$ is equivalent to the standard necessary optimality condition for Problem (1), see [6]. Therefore, whenever $\text{re}(\mathbf{x})$ was used for the Lasso problems, we now use $\|Z(\mathbf{x})\|_\infty$ [including in the step-size rule (11)].

We simulated three instances of the logistic regression problem, whose essential data features are given in Table I; we downloaded the data from the LIBSVM repository <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>, which we refer to for a detailed description of the test problems. In our implementation, the matrix \mathbf{Y} is stored in a column block distributed manner $\mathbf{Y} = [\mathbf{Y}_1 \mathbf{Y}_2 \cdots \mathbf{Y}_P]$, where P is the number of parallel processors. We compared FLEXA $\sigma = 0$ and FLEXA $\sigma = 0.5$ with the other parallel algorithms, namely: FISTA, SpaRSA, and GRock. We do not report results for the sequential methods (GS and ADMM) because we already ascertained that they are not competitive. The tuning of the free parameters in all the algorithms is the same as in Fig. 1 and Fig. 2.

In Fig. 3 we plotted the relative error vs. the CPU time (the latter defined as in Fig. 1 and Fig. 2). Note that this time in order to plot the relative error, we had to preliminary estimate V^* (which we recall is not known for logistic regression problems). In order to do so we ran FLEXA with $\sigma = 0.5$ until the merit function value $\|Z(\mathbf{x}^k)\|_\infty$ went below $1e-6$, and used the corresponding value of the objective function as estimate of V^* . We remark that we used this value only to plot the curves in Fig. 3.

Results on Logistic Regression reinforce the conclusion we made based on the experiments on LASSO problems. Actually, Fig. 3 clearly shows that on these problems both FLEXA methods significantly and consistently outperform all other solution methods.

In conclusion, our experiments indicate that our algorithmic framework can lead to very efficient and practical solution methods for large-scale problems, with the flexibility to adapt to many different problem characteristics.

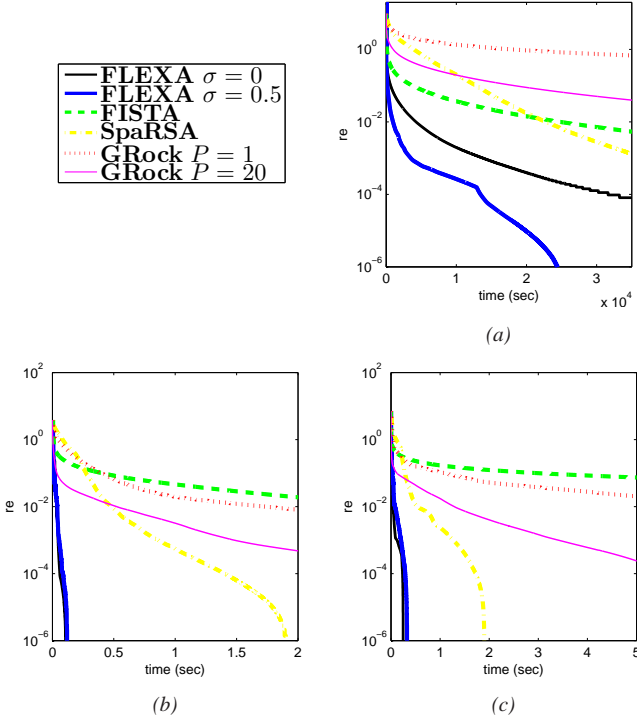


Fig. 3: Relative error vs. time (in seconds) for Logistic Regression: (a) gisette - (b) colon-cancer - (c) leukemia

VII. CONCLUSIONS

We proposed a highly parallelizable algorithmic scheme for the minimization of the sum of a possibly nonconvex differentiable function and a possibly nonsmooth but block-separable convex one. Quite remarkably, our framework leads to different (new) algorithms whose degree of parallelism can be chosen by the user, ranging from fully parallel to sequential schemes, *all of them converging under the same conditions*. Many well know sequential and simultaneous solution methods in the literature are just special cases of our algorithmic framework. Our preliminary tests are very promising, showing that our algorithms consistently outperform state-of-the-art schemes. Experiments on larger and more varied classes of problems (including those listed in Section II) are the subject of our current research. We also plan to investigate asynchronous versions of Algorithm 1, the latter being a very important issue in many distributed settings.

APPENDIX

We first introduce some preliminary results instrumental to prove both Theorem 1 and Theorem 2. Hereafter, for notational simplicity, we will omit the dependence of $\hat{\mathbf{x}}(\mathbf{y}, \tau)$ on τ and write $\hat{\mathbf{x}}(\mathbf{y})$. Given $S \subseteq \mathcal{N}$ and $\mathbf{x} \triangleq (x_i)_{i=1}^N$, we will also denote by $(\mathbf{x})_S$ (or interchangeably \mathbf{x}_S) the vector whose component i is equal to x_i if $i \in S$, and zero otherwise.

A. Intermediate results

Lemma 4: Let $H(\mathbf{x}; \mathbf{y}) \triangleq \sum_i h_i(\mathbf{x}_i; \mathbf{y})$. Then, the following hold:

(i) $H(\bullet; \mathbf{y})$ is uniformly strongly convex on X with constant $c_\tau > 0$, i.e.,

$$(\mathbf{x} - \mathbf{w})^T (\nabla_{\mathbf{x}} H(\mathbf{x}; \mathbf{y}) - \nabla_{\mathbf{x}} H(\mathbf{w}; \mathbf{y})) \geq c_\tau \|\mathbf{x} - \mathbf{w}\|^2, \quad (12)$$

for all $\mathbf{x}, \mathbf{w} \in X$ and given $\mathbf{y} \in X$;

(ii) $\nabla_{\mathbf{x}} H(\mathbf{x}; \bullet)$ is uniformly Lipschitz continuous on X , i.e., there exists a $0 < L_{\nabla H} < \infty$ independent on \mathbf{x} such that

$$\|\nabla_{\mathbf{x}} H(\mathbf{x}; \mathbf{y}) - \nabla_{\mathbf{x}} H(\mathbf{x}; \mathbf{w})\| \leq L_{\nabla H} \|\mathbf{y} - \mathbf{w}\|, \quad (13)$$

for all $\mathbf{y}, \mathbf{w} \in X$ and given $\mathbf{x} \in X$.

Proof: The proof is standard and thus is omitted. ■

Proposition 5: Consider Problem (1) under A1-A6. Then the mapping $X \ni \mathbf{y} \mapsto \hat{\mathbf{x}}(\mathbf{y})$ has the following properties:

(a) $\hat{\mathbf{x}}(\bullet)$ is Lipschitz continuous on X , i.e., there exists a positive constant \hat{L} such that

$$\|\hat{\mathbf{x}}(\mathbf{y}) - \hat{\mathbf{x}}(\mathbf{z})\| \leq \hat{L} \|\mathbf{y} - \mathbf{z}\|, \quad \forall \mathbf{y}, \mathbf{z} \in X; \quad (14)$$

(b) the set of the fixed-points of $\hat{\mathbf{x}}(\bullet)$ coincides with the set of stationary solutions of Problem (1); therefore $\hat{\mathbf{x}}(\bullet)$ has a fixed-point;

(c) for every given $\mathbf{y} \in X$ and for any set $S \subseteq \mathcal{N}$, it holds that

$$\begin{aligned} (\hat{\mathbf{x}}(\mathbf{y}) - \mathbf{y})_S^T \nabla_{\mathbf{x}} F(\mathbf{y})_S + \sum_{i \in S} g_i(\hat{\mathbf{x}}_i(\mathbf{y})) - \sum_{i \in S} g_i(\mathbf{y}_i) \\ \leq -c_\tau \|(\hat{\mathbf{x}}(\mathbf{y}) - \mathbf{y})_S\|^2, \end{aligned} \quad (15)$$

with $c_\tau \triangleq q \min_i \tau_i$.

Proof: We prove the proposition in the following order: (c), (a), (b).

(c): Given $\mathbf{y} \in X$, by definition, each $\hat{\mathbf{x}}_i(\mathbf{y})$ is the unique solution of problem (3); then it is not difficult to see that the following holds: for all $\mathbf{z}_i \in X_i$,

$$(\mathbf{z}_i - \hat{\mathbf{x}}_i(\mathbf{y}))^T \nabla_{\mathbf{x}_i} h_i(\hat{\mathbf{x}}_i(\mathbf{y}); \mathbf{y}) + g_i(\mathbf{z}_i) - g_i(\hat{\mathbf{x}}_i(\mathbf{y})) \geq 0. \quad (16)$$

Summing and subtracting $\nabla_{\mathbf{x}_i} P_i(\mathbf{y}_i; \mathbf{y})$ in (16), choosing $\mathbf{z}_i = \mathbf{y}_i$, and using P2, we get

$$\begin{aligned} (\mathbf{y}_i - \hat{\mathbf{x}}_i(\mathbf{y}))^T (\nabla_{\mathbf{x}_i} P_i(\hat{\mathbf{x}}_i(\mathbf{y}); \mathbf{y}) - \nabla_{\mathbf{x}_i} P_i(\mathbf{y}_i; \mathbf{y})) \\ + (\mathbf{y}_i - \hat{\mathbf{x}}_i(\mathbf{y}))^T \nabla_{\mathbf{x}_i} F(\mathbf{y}) + g_i(\mathbf{y}_i) - g_i(\hat{\mathbf{x}}_i(\mathbf{y})) \\ - \tau_i (\hat{\mathbf{x}}_i(\mathbf{y}) - \mathbf{y}_i)^T \mathbf{Q}_i(\mathbf{y}) (\hat{\mathbf{x}}_i(\mathbf{y}) - \mathbf{y}_i) \geq 0, \end{aligned} \quad (17)$$

for all $i \in \mathcal{N}$. Observing that the term on the first line of (17) is non positive and using P1, we obtain

$$\begin{aligned} (\mathbf{y}_i - \hat{\mathbf{x}}_i(\mathbf{y}))^T \nabla_{\mathbf{x}_i} F(\mathbf{y}) + g_i(\mathbf{y}_i) - g_i(\hat{\mathbf{x}}_i(\mathbf{y})) \\ \geq c_\tau \|\hat{\mathbf{x}}_i(\mathbf{y}) - \mathbf{y}_i\|^2, \end{aligned}$$

for all $i \in \mathcal{N}$. Summing over $i \in S$ we get (15).

(a): We use the notation introduced in Lemma 4. Given $\mathbf{y}, \mathbf{z} \in X$, by optimality and (16), we have, for all \mathbf{v} and \mathbf{w} in X

$$\begin{aligned} (\mathbf{v} - \hat{\mathbf{x}}(\mathbf{y}))^T \nabla_{\mathbf{x}} H(\hat{\mathbf{x}}(\mathbf{y}); \mathbf{y}) + G(\mathbf{v}) - G(\hat{\mathbf{x}}(\mathbf{y})) &\geq 0 \\ (\mathbf{w} - \hat{\mathbf{x}}(\mathbf{z}))^T \nabla_{\mathbf{x}} H(\hat{\mathbf{x}}(\mathbf{z}); \mathbf{z}) + G(\mathbf{w}) - G(\hat{\mathbf{x}}(\mathbf{z})) &\geq 0. \end{aligned}$$

Setting $\mathbf{v} = \hat{\mathbf{x}}(\mathbf{z})$ and $\mathbf{w} = \hat{\mathbf{x}}(\mathbf{y})$, summing the two inequalities above, and adding and subtracting $\nabla_{\mathbf{x}} H(\hat{\mathbf{x}}(\mathbf{y}); \mathbf{z})$, we

obtain:

$$\begin{aligned} & (\widehat{\mathbf{x}}(\mathbf{z}) - \widehat{\mathbf{x}}(\mathbf{y}))^T (\nabla_{\mathbf{x}} H(\widehat{\mathbf{x}}(\mathbf{z}); \mathbf{z}) - \nabla_{\mathbf{x}} H(\widehat{\mathbf{x}}(\mathbf{y}); \mathbf{z})) \\ & \leq (\widehat{\mathbf{x}}(\mathbf{y}) - \widehat{\mathbf{x}}(\mathbf{z}))^T (\nabla_{\mathbf{x}} H(\widehat{\mathbf{x}}(\mathbf{y}); \mathbf{z}) - \nabla_{\mathbf{x}} H(\widehat{\mathbf{x}}(\mathbf{y}); \mathbf{y})). \end{aligned} \quad (18)$$

Using (12) we can lower bound the left-hand-side of (18) as

$$\begin{aligned} & (\widehat{\mathbf{x}}(\mathbf{z}) - \widehat{\mathbf{x}}(\mathbf{y}))^T (\nabla_{\mathbf{x}} H(\widehat{\mathbf{x}}(\mathbf{z}); \mathbf{z}) - \nabla_{\mathbf{x}} H(\widehat{\mathbf{x}}(\mathbf{y}); \mathbf{z})) \\ & \geq c_{\tau} \|\widehat{\mathbf{x}}(\mathbf{z}) - \widehat{\mathbf{x}}(\mathbf{y})\|^2, \end{aligned} \quad (19)$$

whereas the right-hand-side of (18) can be upper bounded as

$$\begin{aligned} & (\widehat{\mathbf{x}}(\mathbf{y}) - \widehat{\mathbf{x}}(\mathbf{z}))^T (\nabla_{\mathbf{x}} H(\widehat{\mathbf{x}}(\mathbf{y}); \mathbf{z}) - \nabla_{\mathbf{x}} H(\widehat{\mathbf{x}}(\mathbf{y}); \mathbf{y})) \\ & \leq L_{\nabla H} \|\widehat{\mathbf{x}}(\mathbf{y}) - \widehat{\mathbf{x}}(\mathbf{z})\| \|\mathbf{y} - \mathbf{z}\|, \end{aligned} \quad (20)$$

where the inequality follows from the Cauchy-Schwartz inequality and (13). Combining (18), (19), and (20), we obtain the desired Lipschitz property of $\widehat{\mathbf{x}}(\bullet)$.

(b): Let $\mathbf{x}^* \in X$ be a fixed point of $\widehat{\mathbf{x}}(\mathbf{y})$, that is $\mathbf{x}^* = \widehat{\mathbf{x}}(\mathbf{x}^*)$. Each $\widehat{\mathbf{x}}_i(\mathbf{y})$ satisfies (16) for any given $\mathbf{y} \in X$. For some $\xi_i \in \partial g_i(\mathbf{x}^*)$, setting $\mathbf{y} = \mathbf{x}^*$ and using $\mathbf{x}^* = \widehat{\mathbf{x}}(\mathbf{x}^*)$ and the convexity of g_i , (16) reduces to

$$(\mathbf{z}_i - \mathbf{x}_i^*)^T (\nabla_{\mathbf{x}_i} F(\mathbf{x}^*) + \xi_i) \geq 0, \quad (21)$$

for all $\mathbf{z}_i \in X_i$ and $i \in \mathcal{N}$. Taking into account the Cartesian structure of X , the separability of G , and summing (21) over $i \in \mathcal{N}$, we obtain $(\mathbf{z} - \mathbf{x}^*)^T (\nabla_{\mathbf{x}} F(\mathbf{x}^*) + \xi) \geq 0$, for all $\mathbf{z} \in X$, with $\mathbf{z} \triangleq (\mathbf{z}_i)_{i=1}^N$ and $\xi \triangleq (\xi_i)_{i=1}^N \in \partial G(\mathbf{x}^*)$; therefore \mathbf{x}^* is a stationary solution of (1).

The converse holds because i) $\widehat{\mathbf{x}}(\mathbf{x}^*)$ is the unique optimal solution of (3) with $\mathbf{y} = \mathbf{x}^*$, and ii) \mathbf{x}^* is also an optimal solution of (3), since it satisfies the minimum principle. ■

Lemma 6: [35, Lemma 3.4, p.121] Let $\{X^k\}$, $\{Y^k\}$, and $\{Z^k\}$ be three sequences of numbers such that $Y^k \geq 0$ for all k . Suppose that

$$X^{k+1} \leq X^k - Y^k + Z^k, \quad \forall k = 0, 1, \dots$$

and $\sum_{k=0}^{\infty} Z^k < \infty$. Then either $X^k \rightarrow -\infty$ or else $\{X^k\}$ converges to a finite value and $\sum_{k=0}^{\infty} Y^k < \infty$.

Lemma 7: Let $\{\mathbf{x}^k\}$ be the sequence generated by Algorithm 1. Then, there is a positive constant \tilde{c} such that the following holds: for all $k \geq 1$,

$$\begin{aligned} & (\nabla_{\mathbf{x}} F(\mathbf{x}^k))^T_{S^k} (\widehat{\mathbf{x}}(\mathbf{x}^k) - \mathbf{x}^k)_{S^k} + \sum_{i \in S^k} g_i(\widehat{\mathbf{x}}_i(\mathbf{x}^k)) \\ & - \sum_{i \in S^k} g_i(\mathbf{x}_i^k) \leq -\tilde{c} \|\widehat{\mathbf{x}}(\mathbf{x}^k) - \mathbf{x}^k\|^2. \end{aligned} \quad (22)$$

Proof: Let j_k be an index in S^k such that $E_{j_k}(\mathbf{x}^k) \geq \rho \max_i E_i(\mathbf{x}^k)$ (Step 3 of Algorithm 1). Then, using the aforementioned bound and (4), it is easy to check that the

following chain of inequalities holds:

$$\begin{aligned} \bar{s}_{j_k} \|\widehat{\mathbf{x}}_{S^k}(\mathbf{x}^k) - \mathbf{x}_{S^k}^k\| & \geq \bar{s}_{j_k} \|\widehat{\mathbf{x}}_{j_k}(\mathbf{x}^k) - \mathbf{x}_{j_k}^k\| \\ & \geq E_{j_k}(\mathbf{x}^k) \\ & \geq \rho \max_i E_i(\mathbf{x}^k) \\ & \geq \left(\rho \min_i \underline{s}_i \right) \left(\max_i \{\|\widehat{\mathbf{x}}_i(\mathbf{x}^k) - \mathbf{x}_i^k\|\} \right) \\ & \geq \left(\frac{\rho \min_i \underline{s}_i}{N} \right) \|\widehat{\mathbf{x}}(\mathbf{x}^k) - \mathbf{x}^k\|. \end{aligned}$$

Hence we have for any k ,

$$\|\widehat{\mathbf{x}}_{S^k}(\mathbf{x}^k) - \mathbf{x}_{S^k}^k\| \geq \left(\frac{\rho \min_i \underline{s}_i}{N \bar{s}_{j_k}} \right) \|\widehat{\mathbf{x}}(\mathbf{x}^k) - \mathbf{x}^k\|. \quad (23)$$

Invoking now Proposition 5(c) with $S = S^k$ and $\mathbf{y} = \mathbf{x}^k$, and using (23), (22) holds true, with $\tilde{c} \triangleq c_{\tau} \left(\frac{\rho \min_i \underline{s}_i}{N \max_j \bar{s}_j} \right)^2$. ■

B. Proof of Theorem 1

We are now ready to prove the theorem. For any given $k \geq 0$, the Descent Lemma [26] yields

$$\begin{aligned} F(\mathbf{x}^{k+1}) & \leq F(\mathbf{x}^k) + \gamma^k \nabla_{\mathbf{x}} F(\mathbf{x}^k)^T (\widehat{\mathbf{z}}^k - \mathbf{x}^k) \\ & \quad + \frac{(\gamma^k)^2 L_{\nabla F}}{2} \|\widehat{\mathbf{z}}^k - \mathbf{x}^k\|^2, \end{aligned} \quad (24)$$

with $\widehat{\mathbf{z}}^k \triangleq (\widehat{\mathbf{z}}_i^k)_{i=1}^N$ and $\mathbf{z}^k \triangleq (\mathbf{z}_i^k)_{i=1}^N$ defined in Step 3 and 4 (Algorithm 1). Observe that

$$\begin{aligned} \|\widehat{\mathbf{z}}^k - \mathbf{x}^k\|^2 & \leq \|\mathbf{z}^k - \mathbf{x}^k\|^2 \\ & \leq 2 \|\widehat{\mathbf{x}}(\mathbf{x}^k) - \mathbf{x}^k\|^2 \\ & \quad + 2 \sum_{i \in \mathcal{N}} \|\mathbf{z}_i^k - \widehat{\mathbf{x}}_i(\mathbf{x}^k)\|^2 \\ & \leq 2 \|\widehat{\mathbf{x}}(\mathbf{x}^k) - \mathbf{x}^k\|^2 + 2 \sum_{i \in \mathcal{N}} (\varepsilon_i^k)^2, \end{aligned} \quad (25)$$

where the first inequality follows from the definition of \mathbf{z}^k and $\widehat{\mathbf{z}}^k$, and in the last inequality we used $\|\mathbf{z}_i^k - \widehat{\mathbf{x}}_i(\mathbf{x}^k)\| \leq \varepsilon_i^k$.

Denoting by \overline{S}^k the complement of S , we also have, for k large enough,

$$\begin{aligned} & \nabla_{\mathbf{x}} F(\mathbf{x}^k)^T (\widehat{\mathbf{z}}^k - \mathbf{x}^k) \\ & = \nabla_{\mathbf{x}} F(\mathbf{x}^k)^T (\widehat{\mathbf{z}}^k - \widehat{\mathbf{x}}(\mathbf{x}^k) + \widehat{\mathbf{x}}(\mathbf{x}^k) - \mathbf{x}^k) \\ & = \nabla_{\mathbf{x}} F(\mathbf{x}^k)^T_{S^k} (\mathbf{z}^k - \widehat{\mathbf{x}}(\mathbf{x}^k))_{S^k} \\ & \quad + \nabla_{\mathbf{x}} F(\mathbf{x}^k)^T_{\overline{S}^k} (\mathbf{x}^k - \widehat{\mathbf{x}}(\mathbf{x}^k))_{\overline{S}^k} \\ & \quad + \nabla_{\mathbf{x}} F(\mathbf{x}^k)^T_{S^k} (\widehat{\mathbf{x}}(\mathbf{x}^k) - \mathbf{x}^k)_{S^k} \\ & \quad + \nabla_{\mathbf{x}} F(\mathbf{x}^k)^T_{\overline{S}^k} (\widehat{\mathbf{x}}(\mathbf{x}^k) - \mathbf{x}^k)_{\overline{S}^k} \\ & = \nabla_{\mathbf{x}} F(\mathbf{x}^k)^T_{S^k} (\mathbf{z}^k - \widehat{\mathbf{x}}(\mathbf{x}^k))_{S^k} \\ & \quad + \nabla_{\mathbf{x}} F(\mathbf{x}^k)^T_{\overline{S}^k} (\widehat{\mathbf{x}}(\mathbf{x}^k) - \mathbf{x}^k)_{\overline{S}^k}, \end{aligned} \quad (26)$$

where in the second equality we used the definition of $\widehat{\mathbf{z}}^k$ and of the set S^k . Now, using (26) and Lemma 7, we can write

$$\begin{aligned}
& \nabla_{\mathbf{x}} F(\mathbf{x}^k)^T (\hat{\mathbf{z}}^k - \mathbf{x}^k) + \sum_{i \in S^k} g_i(\hat{\mathbf{z}}_i^k) - \sum_{i \in S^k} g_i(\mathbf{x}_i^k) \\
&= \nabla_{\mathbf{x}} F(\mathbf{x}^k)^T (\hat{\mathbf{z}}^k - \mathbf{x}^k) + \sum_{i \in S^k} g_i(\hat{\mathbf{x}}_i(\mathbf{x}^k)) \\
&\quad - \sum_{i \in S^k} g_i(\mathbf{x}_i^k) + \sum_{i \in S^k} g_i(\hat{\mathbf{z}}_i^k) - \sum_{i \in S^k} g_i(\hat{\mathbf{x}}_i(\mathbf{x}^k)) \\
&\leq -\tilde{c} \|\hat{\mathbf{x}}(\mathbf{x}^k) - \mathbf{x}^k\|^2 + \sum_{i \in S^k} \varepsilon_i^k \|\nabla_{\mathbf{x}_i} F(\mathbf{x}^k)\| \\
&\quad + L_G \sum_{i \in S^k} \varepsilon_i^k,
\end{aligned} \tag{27}$$

where L_G is a (global) Lipschitz constant for (all) g_i .

Finally, from the definition of $\hat{\mathbf{z}}^k$ and of the set S^k , we have for all k large enough,

$$\begin{aligned}
V(\mathbf{x}^{k+1}) &= F(\mathbf{x}^{k+1}) + \sum_{i \in \mathcal{N}} g_i(\mathbf{x}_i^{k+1}) \\
&= F(\mathbf{x}^{k+1}) + \sum_{i \in \mathcal{N}} g_i(\mathbf{x}_i^k + \gamma^k (\hat{\mathbf{z}}_i^k - \mathbf{x}_i^k)) \\
&\leq F(\mathbf{x}^{k+1}) + \sum_{i \in \mathcal{N}} g_i(\mathbf{x}_i^k) + \gamma^k (\sum_{i \in S^k} (g_i(\hat{\mathbf{z}}_i^k) - g_i(\mathbf{x}_i^k))) \\
&\leq V(\mathbf{x}^k) - \gamma^k (\tilde{c} - \gamma^k L_{\nabla F}) \|\hat{\mathbf{x}}(\mathbf{x}^k) - \mathbf{x}^k\|^2 + T^k,
\end{aligned} \tag{28}$$

where in the first inequality we used the convexity of the g_i 's, whereas the second one follows from (24), (25) and (27), with

$$T^k \triangleq \gamma^k \sum_{i \in S^k} \varepsilon_i^k (L_G + \|\nabla_{\mathbf{x}_i} F(\mathbf{x}^k)\|) + (\gamma^k)^2 L_{\nabla F} \sum_{i \in \mathcal{N}} (\varepsilon_i^k)^2.$$

Using assumption (v), we can bound T^k as

$$T^k \leq (\gamma^k)^2 \left[N\alpha_1(\alpha_2 L_G + 1) + (\gamma^k)^2 L_{\nabla F} (N\alpha_1\alpha_2)^2 \right],$$

which, by assumption (iv) implies $\sum_{k=0}^{\infty} T^k < \infty$. Since $\gamma^k \rightarrow 0$, it follows from (28) that there exist some positive constant β_1 and a sufficiently large k , say \bar{k} , such that

$$V(\mathbf{x}^{k+1}) \leq V(\mathbf{x}^k) - \gamma^k \beta_1 \|\hat{\mathbf{x}}(\mathbf{x}^k) - \mathbf{x}^k\|^2 + T^k, \tag{29}$$

for all $k \geq \bar{k}$. Invoking Lemma 6 with the identifications $X^k = V(\mathbf{x}^{k+1})$, $Y^k = \gamma^k \beta_1 \|\hat{\mathbf{x}}(\mathbf{x}^k) - \mathbf{x}^k\|^2$ and $Z^k = T^k$ while using $\sum_{k=0}^{\infty} T^k < \infty$, we deduce from (29) that either $\{V(\mathbf{x}^k)\} \rightarrow -\infty$ or else $\{V(\mathbf{x}^k)\}$ converges to a finite value and

$$\lim_{k \rightarrow \infty} \sum_{t=\bar{k}}^k \gamma^t \|\hat{\mathbf{x}}(\mathbf{x}^t) - \mathbf{x}^t\|^2 < +\infty. \tag{30}$$

Since V is coercive, $V(\mathbf{x}) \geq \min_{\mathbf{y} \in X} V(\mathbf{y}) > -\infty$, implying that $\{V(\mathbf{x}^k)\}$ is convergent; it follows from (30) and $\sum_{k=0}^{\infty} \gamma^k = \infty$ that $\liminf_{k \rightarrow \infty} \|\hat{\mathbf{x}}(\mathbf{x}^k) - \mathbf{x}^k\| = 0$.

Using Proposition 5, we show next that $\lim_{k \rightarrow \infty} \|\hat{\mathbf{x}}(\mathbf{x}^k) - \mathbf{x}^k\| = 0$; for notational simplicity we will write $\Delta \hat{\mathbf{x}}(\mathbf{x}^k) \triangleq \hat{\mathbf{x}}(\mathbf{x}^k) - \mathbf{x}^k$. Suppose, by contradiction, that $\limsup_{k \rightarrow \infty} \|\Delta \hat{\mathbf{x}}(\mathbf{x}^k)\| > 0$. Then, there exists a $\delta > 0$ such that $\|\Delta \hat{\mathbf{x}}(\mathbf{x}^k)\| > 2\delta$ for infinitely many k and also $\|\Delta \hat{\mathbf{x}}(\mathbf{x}^k)\| < \delta$ for infinitely many k . Therefore, one can always find an infinite set of indexes, say \mathcal{K} , having the following properties: for any $k \in \mathcal{K}$, there exists an integer $i_k > k$ such that

$$\|\Delta \hat{\mathbf{x}}(\mathbf{x}^k)\| < \delta, \quad \|\Delta \hat{\mathbf{x}}(\mathbf{x}^{i_k})\| > 2\delta \tag{31}$$

$$\delta \leq \|\Delta \hat{\mathbf{x}}(\mathbf{x}^j)\| \leq 2\delta \quad k < j < i_k. \tag{32}$$

Given the above bounds, the following holds: for all $k \in \mathcal{K}$,

$$\begin{aligned} \delta &\stackrel{(a)}{<} \|\Delta \hat{\mathbf{x}}(\mathbf{x}^{i_k})\| - \|\Delta \hat{\mathbf{x}}(\mathbf{x}^k)\| \\ &\leq \|\hat{\mathbf{x}}(\mathbf{x}^{i_k}) - \hat{\mathbf{x}}(\mathbf{x}^k)\| + \|\mathbf{x}^{i_k} - \mathbf{x}^k\| \end{aligned} \tag{33}$$

$$\stackrel{(b)}{\leq} (1 + \hat{L}) \|\mathbf{x}^{i_k} - \mathbf{x}^k\| \tag{34}$$

$$\stackrel{(c)}{\leq} (1 + \hat{L}) \sum_{t=k}^{i_k-1} \gamma^t (\|\Delta \hat{\mathbf{x}}(\mathbf{x}^t)_{S^t}\| + \|(\mathbf{z}^t - \hat{\mathbf{x}}(\mathbf{x}^t))_{S^t}\|)$$

$$\stackrel{(d)}{\leq} (1 + \hat{L}) (2\delta + \varepsilon^{\max}) \sum_{t=k}^{i_k-1} \gamma^t, \tag{35}$$

where (a) follows from (31); (b) is due to Proposition 5(a); (c) comes from the triangle inequality, the updating rule of the algorithm and the definition of $\hat{\mathbf{z}}^k$; and in (d) we used (31), (32), and $\|\mathbf{z}^t - \hat{\mathbf{x}}(\mathbf{x}^t)\| \leq \sum_{i \in \mathcal{N}} \varepsilon_i^t$, where $\varepsilon^{\max} \triangleq \max_k \sum_{i \in \mathcal{N}} \varepsilon_i^k < \infty$. It follows from (35) that

$$\liminf_{k \rightarrow \infty} \sum_{t=k}^{i_k-1} \gamma^t \geq \frac{\delta}{(1 + \hat{L})(2\delta + \varepsilon^{\max})} > 0. \tag{36}$$

We show next that (36) is in contradiction with the convergence of $\{V(\mathbf{x}^k)\}$. To do that, we preliminarily prove that, for sufficiently large $k \in \mathcal{K}$, it must be $\|\Delta \hat{\mathbf{x}}(\mathbf{x}^k)\| \geq \delta/2$. Proceeding as in (35), we have: for any given $k \in \mathcal{K}$,

$$\begin{aligned} \|\Delta \hat{\mathbf{x}}(\mathbf{x}^{k+1})\| - \|\Delta \hat{\mathbf{x}}(\mathbf{x}^k)\| &\leq (1 + \hat{L}) \|\mathbf{x}^{k+1} - \mathbf{x}^k\| \\ &\leq (1 + \hat{L}) \gamma^k (\|\Delta \hat{\mathbf{x}}(\mathbf{x}^k)\| + \varepsilon^{\max}). \end{aligned}$$

It turns out that for sufficiently large $k \in \mathcal{K}$ so that $(1 + \hat{L}) \gamma^k < \delta/(\delta + 2\varepsilon^{\max})$, it must be

$$\|\Delta \hat{\mathbf{x}}(\mathbf{x}^k)\| \geq \delta/2; \tag{37}$$

otherwise the condition $\|\Delta \hat{\mathbf{x}}(\mathbf{x}^{k+1})\| \geq \delta$ would be violated [cf. (32)]. Hereafter we assume without loss of generality that (37) holds for all $k \in \mathcal{K}$ (in fact, one can always restrict $\{\mathbf{x}^k\}_{k \in \mathcal{K}}$ to a proper subsequence).

We can show now that (36) is in contradiction with the convergence of $\{V(\mathbf{x}^k)\}$. Using (29) (possibly over a subsequence), we have: for sufficiently large $k \in \mathcal{K}$,

$$\begin{aligned} V(\mathbf{x}^{i_k}) &\leq V(\mathbf{x}^k) - \beta_2 \sum_{t=k}^{i_k-1} \gamma^t \|\Delta \hat{\mathbf{x}}(\mathbf{x}^t)\|^2 + \sum_{t=k}^{i_k-1} T^t \\ &\stackrel{(a)}{<} V(\mathbf{x}^k) - \beta_2 (\delta^2/4) \sum_{t=k}^{i_k-1} \gamma^t + \sum_{t=k}^{i_k-1} T^t \end{aligned} \tag{38}$$

where in (a) we used (32) and (37), and β_2 is some positive constant. Since $\{V(\mathbf{x}^k)\}$ converges and $\sum_{k=0}^{\infty} T^k < \infty$, (38) implies $\lim_{k \rightarrow \infty} \sum_{t=k}^{i_k-1} \gamma^t = 0$, which contradicts (36).

Finally, since the sequence $\{\mathbf{x}^k\}$ is bounded [due to the coercivity of V and the convergence of $\{V(\mathbf{x}^k)\}$], it has at least one limit point $\bar{\mathbf{x}}$ that must belong to X . By the continuity of $\hat{\mathbf{x}}(\bullet)$ [Proposition 5(a)] and $\lim_{k \rightarrow \infty} \|\hat{\mathbf{x}}(\mathbf{x}^k) - \mathbf{x}^k\| = 0$, it must be $\hat{\mathbf{x}}(\bar{\mathbf{x}}) = \bar{\mathbf{x}}$. By Proposition 5(b) $\bar{\mathbf{x}}$ is also a stationary solution of Problem (1).

As a final remark, note that if $\varepsilon_i^k = 0$ for every i and for every k large enough, i.e., if eventually $\hat{\mathbf{x}}(\mathbf{x}^k)$ is computed exactly, there is no need to assume that G is globally Lipschitz.

In fact in (27) the term containing L_G disappears, and actually all the terms T^k are zero and all the subsequent derivations independent of the Lipschitzianity of G . \square

C. Proof of Theorem 2

We show next that Algorithm 2 is just an instance of the inexact Jacobi scheme described in Algorithm 1 satisfying the convergence conditions in Theorem 1; which proves Theorem 2. It is not difficult to see that this boils down to proving that, for all $p \in P$ and $i \in I_p$, the sequence z_{pi}^k in Step 2a) of Algorithm 2 satisfies

$$\|z_{pi}^k - \hat{x}_{pi}(\mathbf{x}^k)\| \leq \tilde{\varepsilon}_{pi}^k, \quad (39)$$

for some $\{\tilde{\varepsilon}_{pi}^k\}$ such that $\sum_n \tilde{\varepsilon}_{pi}^k \gamma^k < \infty$. The following holds for the LHS of (39):

$$\begin{aligned} \|z_{pi}^k - \hat{x}_{pi}(\mathbf{x}^k)\| &\leq \|\hat{x}_{pi}(\mathbf{x}_{pi<}^{k+1}, \mathbf{x}_{pi\geq}^k, \mathbf{x}_{-p}) - \hat{x}_{pi}(\mathbf{x}^k)\| \\ &\quad + \|z_{pi}^k - \hat{x}_{pi}(\mathbf{x}_{pi<}^{k+1}, \mathbf{x}_{pi\geq}^k, \mathbf{x}_{-p})\| \\ &\stackrel{(a)}{\leq} \|\hat{x}_{pi}(\mathbf{x}_{pi<}^{k+1}, \mathbf{x}_{pi\geq}^k, \mathbf{x}_{-p}) - \hat{x}_{pi}(\mathbf{x}^k)\| + \varepsilon_{pi}^k \\ &\stackrel{(b)}{\leq} \hat{L} \|\mathbf{x}_{pi<}^{k+1} - \mathbf{x}_{pi<}^k\| + \varepsilon_{pi}^k \\ &\stackrel{(c)}{=} \hat{L} \gamma^k \|(\mathbf{z}_{pi<}^k - \mathbf{x}_{pi<}^k)\| + \varepsilon_{pi}^k \\ &\leq \hat{L} \gamma^k \left(\sum_{j=1}^{i-1} (\|\mathbf{z}_{pj}^k - \hat{x}_{pj}(\mathbf{x}^k)\| + \|\hat{x}_{pj}(\mathbf{x}^k) - \mathbf{x}_{pj}^k\|) \right) \\ &\quad + \varepsilon_{pi}^k \\ &\stackrel{(d)}{\leq} \hat{L} \gamma^k \beta_i + \hat{L} \gamma^k \sum_{j<i} \varepsilon_{pj}^k + \varepsilon_{pi}^k, \end{aligned}$$

where (a) follows from the error bound in Step 2a) of Algorithm 2; in (b) we used Proposition 5a); (c) follows from Step 2b); and in (d) we used induction, where $\beta_i < \infty$ is a positive constant. It turns out that (39) is satisfied choosing $\tilde{\varepsilon}_{pi}^k \triangleq \hat{L} \gamma^k \beta_i + \hat{L} \gamma^k \sum_{j<i} \varepsilon_{pj}^k + \varepsilon_{pi}^k$. \square

REFERENCES

- [1] F. Facchinei, S. Sagratella, and G. Scutari, "Flexible parallel algorithms for big data optimization," in *Proc. of the IEEE 2014 International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2014)*, Florence, Italy, May 4-9, [Online]. Available: <http://arxiv.org/abs/1311.2444>
- [2] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.
- [3] Z. Qin, K. Scheinberg, and D. Goldfarb, "Efficient block-coordinate descent algorithms for the group lasso," *Mathematical Programming Computation*, vol. 5, pp. 143–169, June 2013.
- [4] A. Rakotomamonjy, "Surveying and comparing simultaneous sparse approximation (or group-lasso) algorithms," *Signal processing*, vol. 91, no. 7, pp. 1505–1526, July 2011.
- [5] G.-X. Yuan, K.-W. Chang, C.-J. Hsieh, and C.-J. Lin, "A comparison of optimization methods and software for large-scale l1-regularized linear classification," *The Journal of Machine Learning Research*, vol. 9999, pp. 3183–3234, 2010.
- [6] R. H. Byrd, J. Nocedal, and F. Oztoprak, "An Inexact Successive Quadratic Approximation Method for Convex L-1 Regularized Optimization," *arXiv preprint arXiv:1309.3529*, 2013.
- [7] K. Fountoulakis and J. Gondzio, "A Second-Order Method for Strongly Convex L1-Regularization Problems," *arXiv preprint arXiv:1306.5386*, 2013.
- [8] Y. Nesterov, "Efficiency of coordinate descent methods on huge-scale optimization problems," *SIAM Journal on Optimization*, vol. 22, no. 2, pp. 341–362, 2012.
- [9] I. Necoara and D. Clipici, "Efficient parallel coordinate descent algorithm for convex optimization problems with separable constraints: application to distributed MPC," *Journal of Process Control*, vol. 23, no. 3, pp. 243–253, March 2013.
- [10] Y. Nesterov, "Gradient methods for minimizing composite functions," *Mathematical Programming*, vol. 140, pp. 125–161, August 2013.
- [11] P. Tseng and S. Yun, "A coordinate gradient descent method for nonsmooth separable minimization," *Mathematical Programming*, vol. 117, no. 1–2, pp. 387–423, March 2009.
- [12] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM Journal on Imaging Sciences*, vol. 2, no. 1, pp. 183–202, Jan.
- [13] S. J. Wright, R. D. Nowak, and M. A. Figueiredo, "Sparse reconstruction by separable approximation," *IEEE Trans. on Signal Processing*, vol. 57, no. 7, pp. 2479–2493, July 2009.
- [14] J. K. Bradley, A. Kyröla, D. Bickson, and C. Guestrin, "Parallel coordinate descent for l1-regularized loss minimization," in *Proc. of the 28th International Conference on Machine Learning*, Bellevue, WA, USA, June 28–July 2, 2011.
- [15] Z. Yin, P. Ming, and Y. Wotao, "Parallel and Distributed Sparse Optimization," 2013. [Online]. Available: [http://www.caam.rice.edu/\\$\sim\\$optimization/disparse/](http://www.caam.rice.edu/\simoptimization/disparse/)
- [16] P. Richtárik and M. Takáč, "Parallel coordinate descent methods for big data optimization," *arXiv preprint arXiv:1212.0873*, 2012.
- [17] M. Razaviyayn, M. Hong, and Z.-Q. Luo, "A unified convergence analysis of block successive minimization methods for nonsmooth optimization," *SIAM Journal on Optimization*, vol. 23, no. 2, pp. 1126–1153, 2013.
- [18] P. L. Bühlmann, S. A. van de Geer, and S. Van de Geer, *Statistics for high-dimensional data*. Springer, 2011.
- [19] S. Sra, S. Nowozin, and S. J. Wright, Eds., *Optimization for Machine Learning*, ser. Neural Information Processing. Cambridge, Massachusetts: The MIT Press, Sept. 2011.
- [20] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski, *Optimization with Sparsity-inducing Penalties*. Foundations and Trends® in Machine Learning, Now Publishers Inc, Dec. 2011.
- [21] G. Scutari, F. Facchinei, P. Song, D. Palomar, and J.-S. Pang, "Decomposition by Partial linearization: Parallel optimization of multi-agent systems," *IEEE Trans. on Signal Processing*, vol. 62, pp. 641–656, Feb. 2014.
- [22] M. Yuan and Y. Lin, "Model selection and estimation in regression with grouped variables," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 68, no. 1, pp. 49–67, 2006.
- [23] S. K. Shevade and S. S. Keerthi, "A simple and efficient algorithm for gene selection using sparse logistic regression," *Bioinformatics*, vol. 19, no. 17, pp. 2246–2253, 2003.
- [24] L. Meier, S. Van De Geer, and P. Bühlmann, "The group lasso for logistic regression," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 70, no. 1, pp. 53–71, 2008.
- [25] D. Goldfarb, S. Ma, and K. Scheinberg, "Fast alternating linearization methods for minimizing the sum of two convex functions," *Mathematical Programming*, vol. 141, pp. 349–382, Oct. 2013.
- [26] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, 2nd ed. Athena Scientific Press, 1989.
- [27] F. Facchinei and J.-S. Pang, *Finite-Dimensional Variational Inequalities and Complementarity Problem*. Springer-Verlag, New York, 2003.
- [28] G. Cohen, "Optimization by decomposition and coordination: A unified approach," *IEEE Trans. on Automatic Control*, vol. 23, no. 2, pp. 222–232, April 1978.
- [29] —, "Auxiliary problem principle and decomposition of optimization problems," *Journal of Optimization Theory and Applications*, vol. 32, no. 3, pp. 277–305, Nov. 1980.
- [30] M. Patriksson, "Cost approximation: a unified framework of descent algorithms for nonlinear programs," *SIAM Journal on Optimization*, vol. 8, no. 2, pp. 561–582, 1998.
- [31] M. Fukushima and H. Mine, "A generalized proximal point algorithm for certain non-convex minimization problems," *International Journal of Systems Science*, vol. 12, no. 8, pp. 989–1000, 1981.
- [32] H. Mine and M. Fukushima, "A minimization method for the sum of a convex function and a continuously differentiable function," *Journal of Optimization Theory and Applications*, vol. 33, no. 1, pp. 9–23, Jan. 1981.
- [33] Y. Saad, *Numerical methods for large eigenvalue problems*, ser. Classics in Applied Mathematics (Book 66). SIAM–Society for Industrial & Applied Mathematics; Revised edition, May 2011, vol. 158.
- [34] Z.-Q. Luo and M. Hong, "On the linear convergence of the alternating direction method of multipliers," *arXiv preprint arXiv:1208.3922*, 2012.
- [35] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Cambridge, Massachusetts: Athena Scientific Press, May. 2011.